Rippl User's Manual and Reference

Kevin N. Smith

UUCS-89-014

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

May 1989

RIPPL USER'S MANUAL AND REFERENCE

by

Kevin N. Smith

Techical Report
UUCS-89-014

The University of Utah

May 1989

CONTENTS

CHAPTERS

CHAPTER 1

INTRODUCTION

This manual assumes a knowledge of the RIPPL project. If the reader if unfamiliar with it, they are encouraged to read Kevin Smith's Master's thesis titled: "Restructurable Interconnect of Path Programmable Logic." It is available in the Marriot Library at the University of Utah.

CHAPTER 2

SOFTWARE TOOLS

In order for RIPPL to be a feasible design methodology, a complete set of software tools were integrated together. The existing PPL tool suite consists primarily of a tile placer (TILER), a circuit extractor (SIMPPLEX), a circuit simulator (SIMPPL), and a CIF generator (PPL2CIF). These tools are also used with RIPPL. There was a need for two new tools to be built:

1. A cell editor (CELLED) with which the RIPPL cell set could be designed
2. A link/cut list generator (MakeZap)

2.1 CELLED

The first tool designed was a cell editor called "CELLED". This software package is used to create the cells for the RIPPL cell set. Actually, two programs written: CELLED which is used for editing nontestable cells and CELLED_TEST which is used for editing testable cells. Since the only difference between these programs is the data used to define the circuits, only CELLED will be described.

Having a cell editor is a very powerful concept. It enables the designer to have an unlimited cell set. If there is a certain RIPPL cell desired but not available in the current cell set, it can easily be designed and added to the cell set without doing any layout. There is no need to refabricate anything. A new laser link/cut list is all that is needed.

CELLED is a user friendly, simple, graphics oriented software package. It facilitates the designing of a RIPPL cell set. CELLED was written in C on an

HP3000. The graphics was done with the X-windows tool kit. Doing this enables the software to be a portable as possible. X-windows is fast becoming the industry standard along with C. X-windows was designed specifically to work with programs written in C. Doing the programming in C allows the program to run very fast.

The basic use of CELLED will first be explained, including its features and functions. After this, a detailed explanation of the programs data structures and algorithms will be given.

2.1.1 Starting up CELLED

In order to run the CELLED program, X-windows must be running on the terminal. CELLED can the be invoked in 4 different ways. These are:

1. "celled",
2. "celled n m",
3. "celled FileName", and
4. "celled n m FileName".

If the user simply types "celled", then the cell to be edited is assumed to be of size 1 x 1. This cell can be modified. It is assumed to have no name. When the user wishes to save the cell, a name will be queried.

If the user types "celled m n" where m and n are positive integers, then a m-row, n-column cell is created. The name of the cell will be queried when doing a save.

If the user types "celled FileName", this specifies FileName as the cell to be edited. If FileName does not exist, it is created. The file will contain the size of the cell.

If the user types "celled m n FileName", this specifies FileName as the cell to be edited and the size to be m x n. If FileName already exists, the size in the file must equal that of m and n. If FileName does not exist, it is created.

The initial display is shown in Figure 2.1. There are three main windows: the menu window, the message window, and the drawing window. If a larger cell were typed in, the initial display will always start off showing the entire m x n cell.

The menu window contains all of the possible commands. These include zooming, editing, generating a print file, generating a CIF file, generating a link/cut file, saving the CELLED file, and quitting. The message window is used by CELLED to post important messages to the user. These include warnings and basic feed back. CELLED also uses dialog boxes to communicate. The drawing window is a bit of a misnomer. The user never actually draws anything. CELLED draws the circuit in the drawing window. The user edits the cell in the drawing window. This editing is merely defining breaks and cuts. All functions in the menu window are executed by clicking any mouse button down in the menu item.

All CELLED windows will automatically be redrawn whenever it is raised above other windows.

2.1.2 Commands

There are two main command modes: editing and zooming. The user can toggle between these two modes. All other commands are done as subcommands when in either the edit mode or the zoom mode. The menu items reflect which mode the user is in based on the shading of the item. In the zoom mode, the zoom menu item is dark. In the edit mode, the edit menu item is dark. The menu item commands are shown in Figure 2.2.

2.1.2.1 Editing

The editing function is used to define which links and cuts are made. When CELLED begins, it defaults to the edit mode. Potential cuts which are not yet made are displayed as a dashed "X". A solid "X" means that the cut is made. A

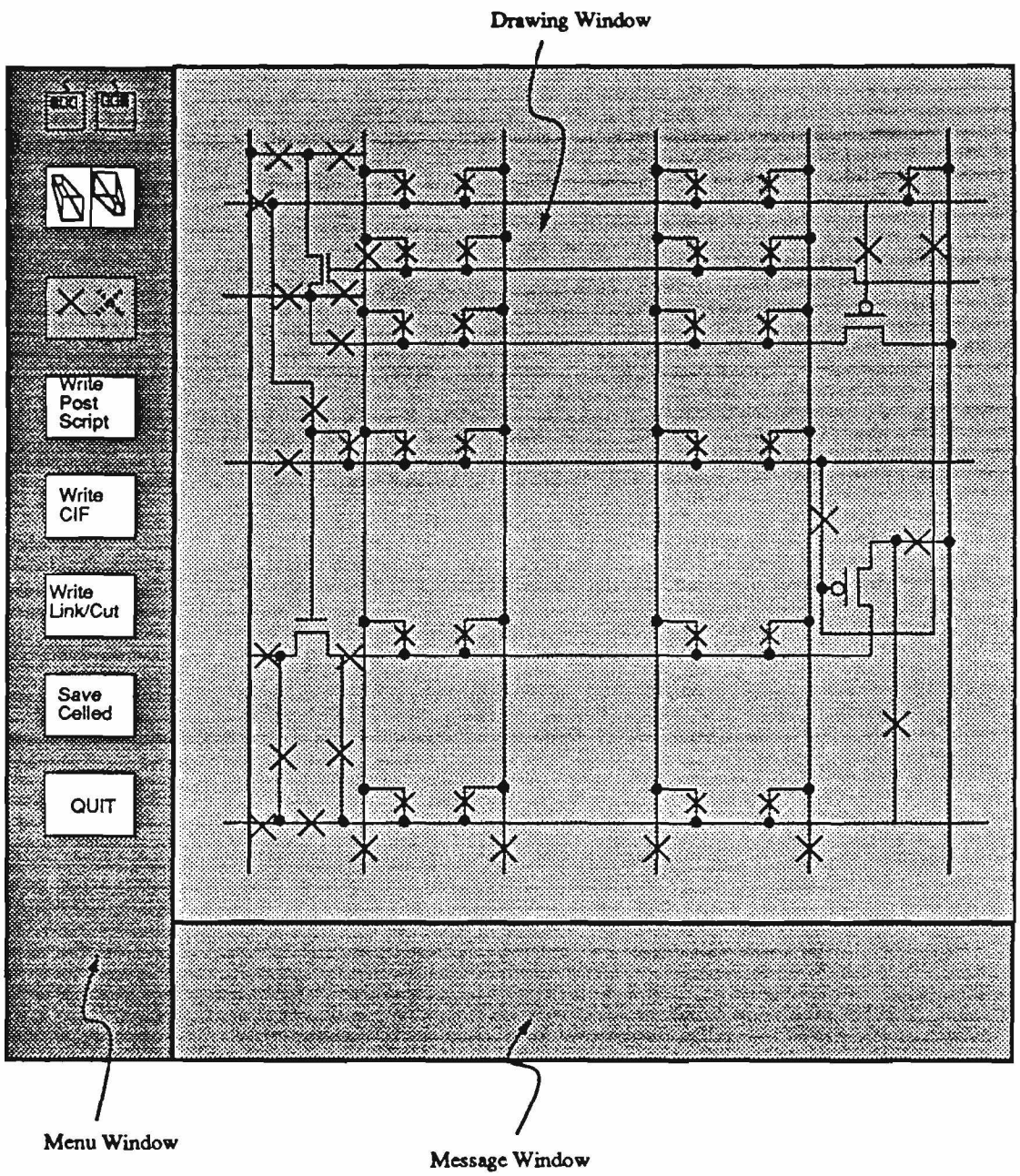


Figure 2.1. The start up screen for a 1x1 cell

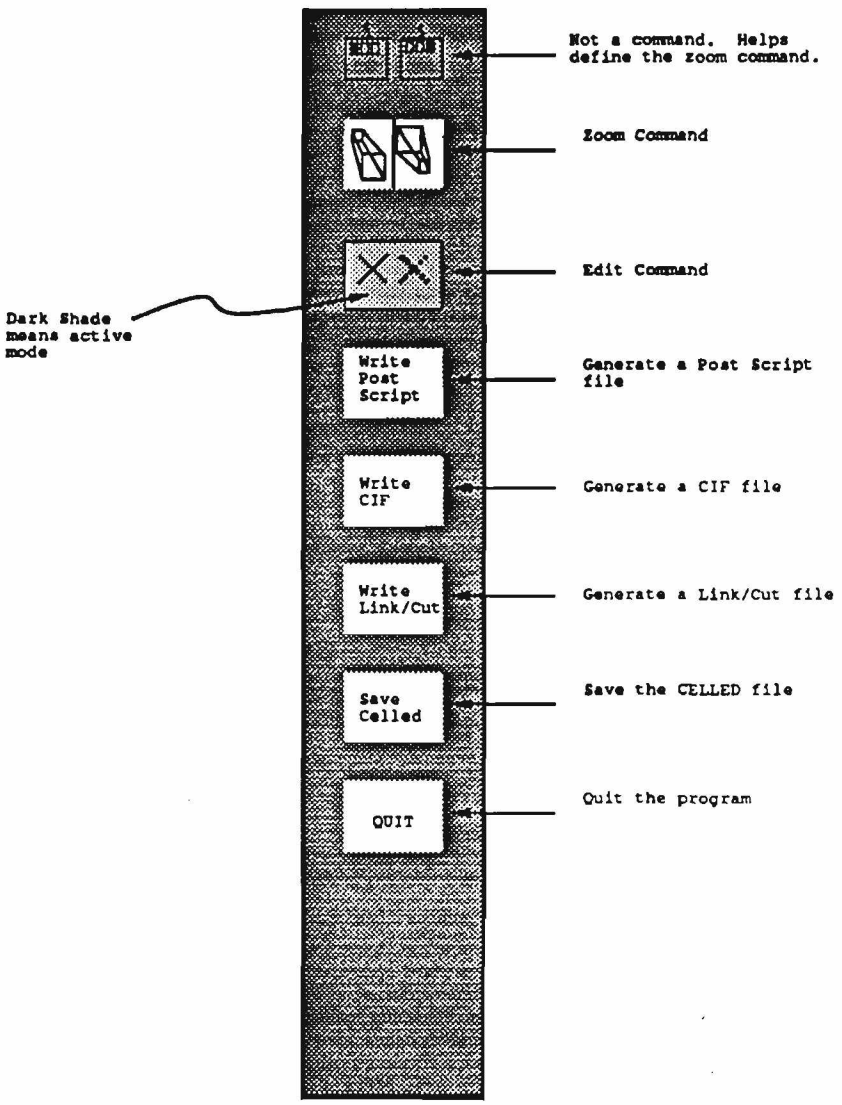


Figure 2.2. The menu commands

empty square is a potential link. A solid square means that the link is made. The editing is done by clicking any mouse button down near a link or cut. This will toggle the link or cut between being solid or dashed.

2.1.2.2 Zooming

When editing complex cells, it becomes difficult to view the entire cell displayed on the screen. For this reason, a zoom function was implemented.

The zoom function is the only function which uses the mouse buttons differently depending on which one was pressed. This is shown in the menu. The left button is used to "zoom in" while the right button is used to "zoom out."

"Zooming in" is done by clicking the left button on the position of the circuit which is to become the new center of the display. The circuit will be redrawn, scaled twice as close with the mouse click position being the center point. This actually does two functions, a transformation and a scale.

"Zooming out" is done by clicking anywhere in the drawing window. The circuit is scaled and transformed back to the previous scale. The circuit cannot be zoomed out beyond the original display size.

2.1.2.3 Write Post Script

All write functions are performed in the same way. When the user clicks on this command, a dialog box is displayed on the screen. It is through this dialog box that the file name is set up. If a name was originally typed in, that name will be shown in the dialog box. This name can be edited by backspacing and typing. When the name is correct, either the save or cancel button must be clicked in. Typing a carriage return is the same as clicking in the save button. This dialog box is shown is Figure 2.3. A note is displayed in the message box stating that a ".prt" extension is automatically appended to the file name.

The print file generated can be printed on any post script printer.

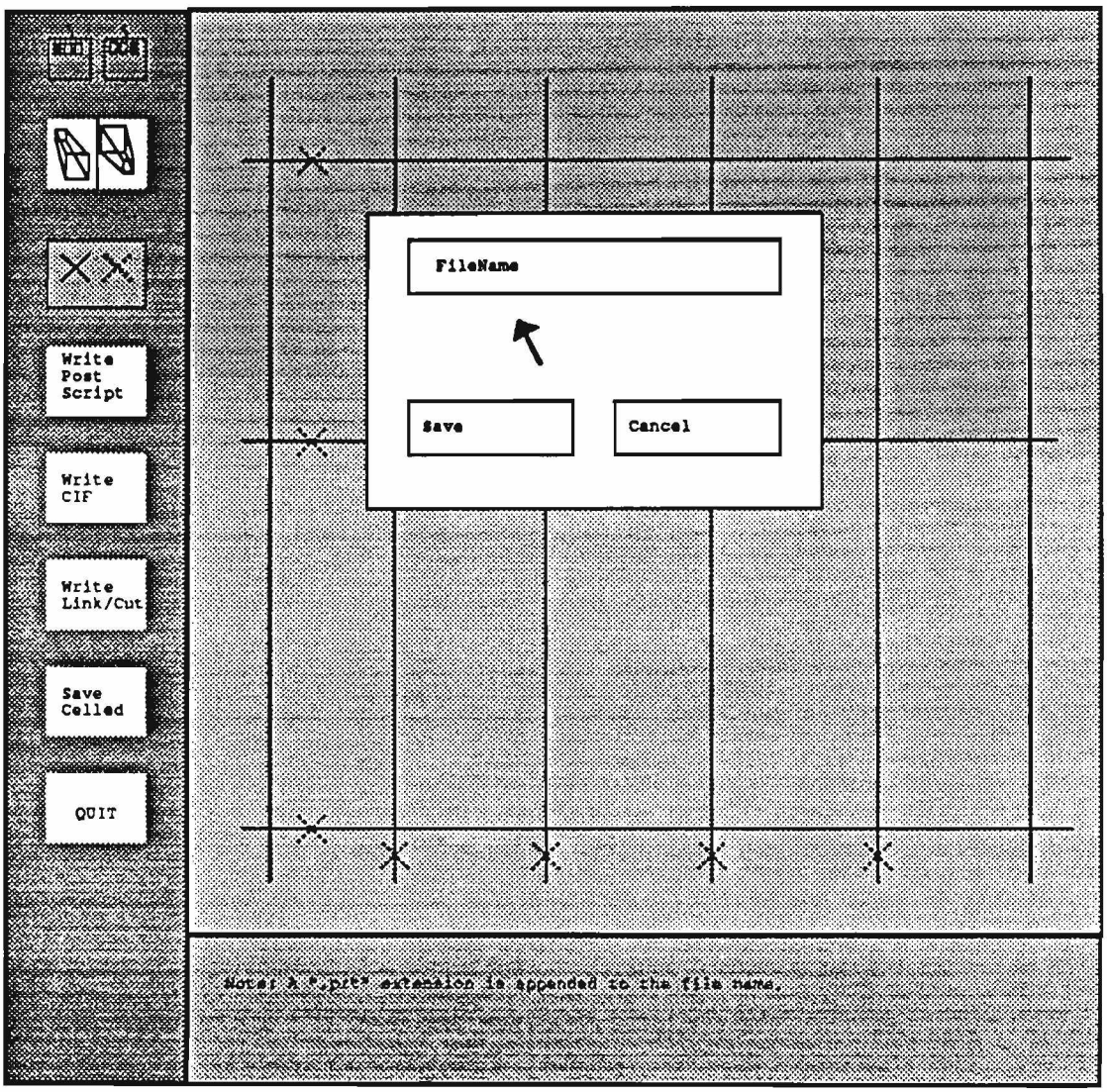


Figure 2.3. The write/save dialog box

2.1.2.4 Write CIF

This function generates a CIF file which corresponds to the circuit. A ".cif" extension is appended to the file name.

The main function of this CIF file is simulation. A number of different approaches were taken for simulating a CELLED file. One approach was to generate a SPICE file or a SIM file directly from CELLED. The problem encountered with a simulation file was that sometimes nodes in the circuit exist but are not used. This is the case with a column or row wire which simply passes through the circuit. SPICE does not allow a floating node. It must be involved in the circuit. In order to generate a usable SPICE file, all floating nodes would have to be removed. This presents a large graph problem. The same problem exists for a SIM file.

The VLSI Technology Incorporated (VTI) design tools are available at the University of Utah. These tools have the capability of extracting a SIM file and a SPICE file from a CIF file. It was decided that it would be most efficient to generate a CIF file for each CELLED cell and then generate the SIM file using VTI. The VTI tools also have the advantage of easily modeling a cut. There is a special CIF layer called "exclude". When this layer is placed, all geometry underneath it is ignored. This is the perfect way to model a cut. Whenever a cell has a cut, a 4 x 4 μ box of exclude is placed in the CIF file. Links are modeled as 4 x 4 μ boxes of diffusion. This causes a connection between the two areas of diffusion where the link would take place.

2.1.2.5 Write Link/Cut

This command generates a Link/Cut file. This file contains the locations of each link or cut to be made. A ".cut" extension is appended to the file name. The format is given in Figure 2.4.

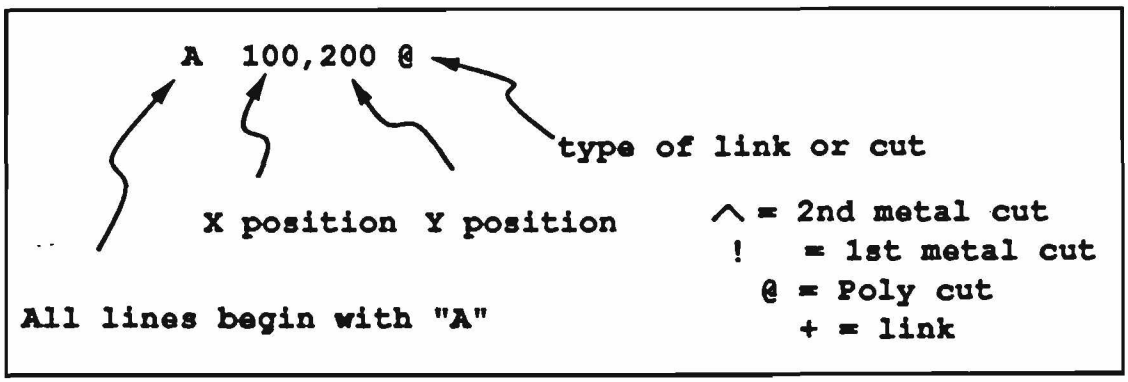


Figure 2.4. The format of the link/cut list

2.1.2.6 Save Celled

This command saves out a CELLED file with a ".cel" extension. This is the file which is read into CELLED when starting the program. It defines all of the links and cuts, the size of the cell, and includes a special header.

2.1.2.7 Quit

When quitting, a dialog box is presented. If there have been changes made since the last save, a warning is presented. The quit command can be canceled or confirmed. If no changes were made, a confirmation dialog box is presented, keeping the user from accidentally exiting.

2.1.3 Data Structures and Algorithms

The CELLED program was written in C. The compiled program occupies 221K of memory. It is approximately 4000 lines of code. About 15 percent of the code is comments.

The main data structures used in CELLED are shown in Figure 2.5. The data structure used to contain all information pertaining to links and cuts is a "m x n" array called LinksAndCuts. This is an array of cells. A cell is a structure containing

four arrays. There is one array for each type of link and cut. These arrays contain either a 1 or 0, depending whether or not the link or cut has been made.

The LinksAndCuts array must be dynamically allocated. Its size corresponds to the cell size specified by the user. The arrays in a cell are all a fixed size. These sizes correspond to the number of links and cuts in the physical layout of the base cell.

Another main data structure, named MainLinksAndCuts, is used to maintain the information corresponding to the drawing positions of the links and cuts. It contains four arrays of LinkAndCutLocations. A LinkAndCutLocation is a structure with two fields: an x position and a y position. These correspond to the drawn position of the link or cut.

This program is event driven, meaning that it only reacts to stimuli presented in the form of events. An event can be generated by a mouse click, exposing of a window, or typing at the key board.

All code shown in this thesis will be given as pseudo code. Pseudo code resembles C but is much simpler. Comments are denoted by stars "*".

2.1.3.1 Initialization

The main function of CELLED is shown in Figure 2.6. This function has an initialization phase. This is detailed in Figure 2.7 through Figure 2.10.

The function DoInitialIO function processes the command line. As described earlier, CELLED can be invoked in four different ways. Each is covered with the initial SWITCH statement. When allocating the LinksAndCuts "n x m" array, the scale factors to be used to display the circuit are calculated. The center point also must be calculated. This is simply the center of a 1 x 1 cell multiplied by the number of cells. This enables the display to start up displaying all of the circuit.

The file processing functions are shown in Figure 2.8. The ProcessFileName function is used to process the file name given at the command line. A CELLED

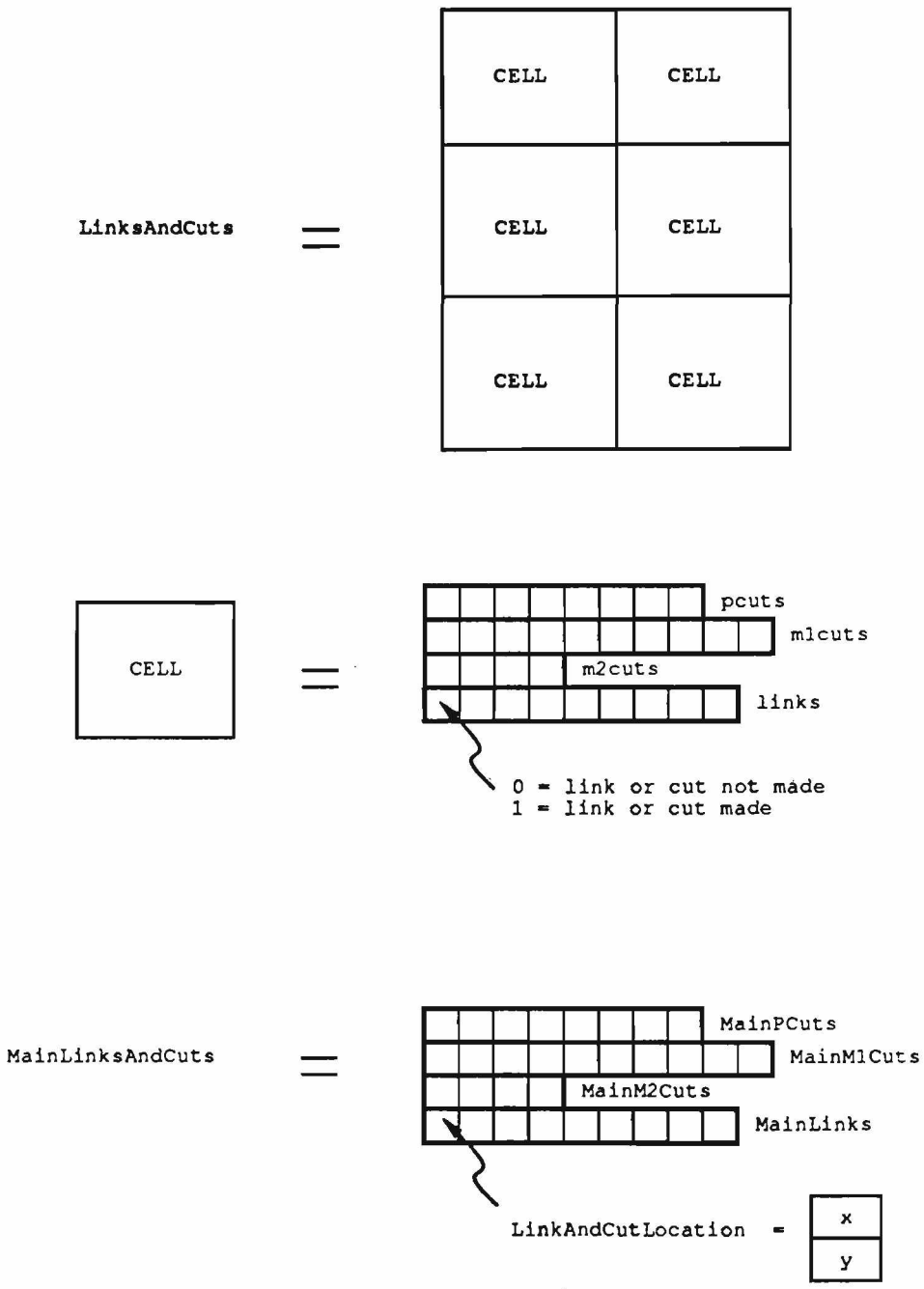


Figure 2.5. The main data structures used in CELLED

```
FUNC MAIN

   ********************
   * Initialization *
   ********************

   DoInitialIO;
   Initialize;

   ********************
   * Enter edit mode *
   ********************
   DoEdit;

   LOOP
     Sleep Until Event;
     SWITCH (event)
        CASE Mouse Down Event in Edit Menu:
             DoEdit;
        CASE Mouse Down Event in Zoom Menu:
             DoZoom;
        CASE Expose Event:
             DoExpose;
   END LOOP

ENDFUNC
```

Figure 2.6. The main function

```
FUNC DoInitialIO

   SWITCH (number of arguments given at command line)
        CASE (1 given):
             Allocate a 1x1 cell;
        ************************************************
        * When a n x m cell is created, all of the scale *
        * factors are calculated and stored for use in   *
        * in displaying the circuit.                     *
        ************************************************
        CASE (2 given):
             ProcessFileName(2nd arg);
        CASE (3 given):
             Allocate an m x n cell;
        CASE (4 given):
             Allocate an m x n cell;
             ProcessFileName(2nd arg);

ENDFUNC
```

Figure 2.7. The initialization I/O functions

file has a special format. This format is shown in Figure 2.9. The format of the file to be read in must follow this format.

Errors in this file are handled in different ways. If the header is bad, then a 1 x 1 cell is opened with no links or cuts. Any saves will write over the file given in the command line. This is noted in the message window.

If the sizes in the file do not correspond to those typed in, the dimensions given in the file are the ones used. A message reflecting this is shown in the message window.

If there is bad data in the link/cut list (i.e., if there were a 2 instead of a 1 or 0), then all the data in the link/cut list is assumed bad. All links and cuts are initialized to 0, i.e. no links or cuts.

The Initialize function handles the X-window set up. This includes defining the size of windows, colors, and the contents of each window.

```
FUNC ProcessFileName

   ******************************************************
   * A file name can be typed in a FileName.cel         *
   *  or just FileName and the ".cel" will be assumed.  *
   ******************************************************

   IF file name does not have an extension
     THEN assume it to be ".cel";

   IF file does not exist
     THEN initialize all cuts and links as FALSE
     ELSE
         check accessibility of file;
         ReadInFile;

ENDFUNC

FUNC ReadInFile

   Check for identifying header;
   Check dimensions against those typed in;
   Check data in file for validity;
        ******************************
        * Each link and cut can only *
        * have values of 1 or 0      *
        ******************************

ENDFUNC
```

Figure 2.8. The file functions

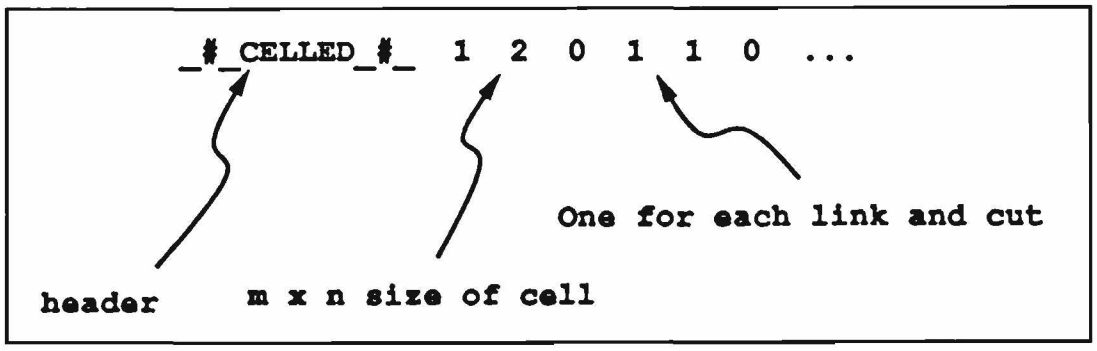


Figure 2.9. The format of the ".cel" file

The drawing of the circuit on the screen was tried two different ways. It was hoped that one method might would paint the screen faster the the other. The first involved the use of a number of "line" calls and the second involved the use of one large "polyline" with some undrawn vertices. It was interesting to note that both methods took approximately the same amount of time to display a circuit. The "line" method was employed in the CELLED program and the "polyline" method was used in the CELLED_TEST program.

A single copy of the main base cell circuit definition exists. This copy includes all of the information necessary for displaying the base circuit i.e., line positions, circle positions, line thickness. To display a "m x n" cell, CELLED simply makes function calls to draw the base circuit at a offset position.

2.1.3.2 Editing

After doing the initialization functions, the program then enters the edit mode. This corresponds to the function DoEdit. There are two modes of the program: edit mode and zoom mode. While in each of these modes, all other commands can be done as subcommands, but the program can never simultaneously be in both modes. This concept is shown in Figure 2.11.

```
FUNC Initialize

   Set up the positions of the cuts and links;
   Set up some standard pixmap colors;
   Create Windows;
   Display Windows;

ENDFUNC
```

Figure 2.10. The initialize function

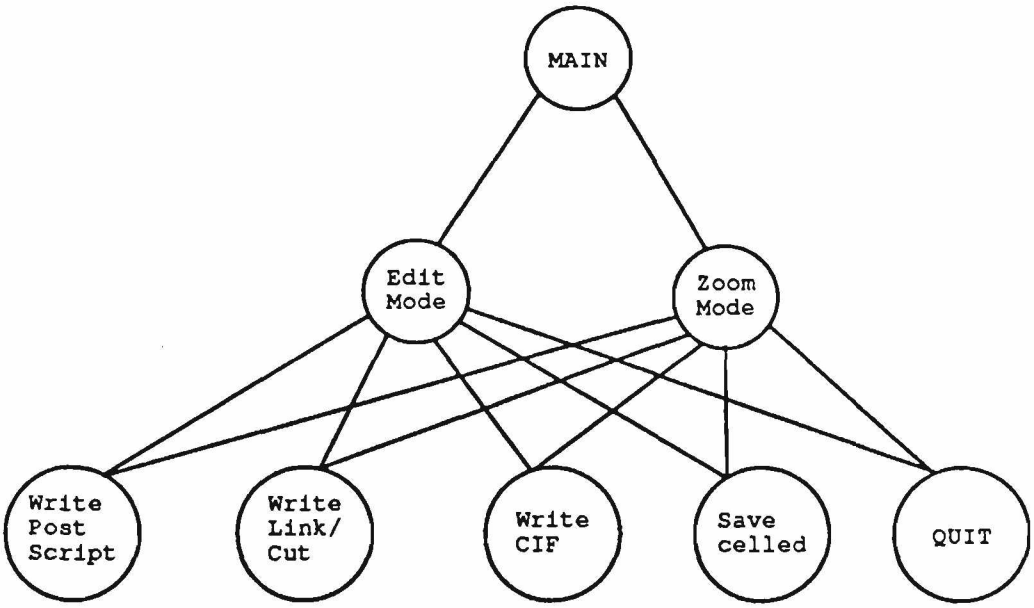


Figure 2.11. The hierarchy of the main commands

The DoEdit function (shown in Figure 2.12) is a loop which branches to the appropriate function based on the next event. The only way to leave this loop is to either QUIT or change to ZOOM. The function unique to this loop is that of editing. If the mouse button is clicked in the draw window, it is checked to see if it was near a link or cut. The user does not have to click exactly on the link or cut. This is difficult since the lines are only one pixel thick. In order to remedy this, a link or cut is defined as occupying a square area defined by its outer most points. Clicking anywhere in this area is considered clicking on the link or cut.

Each time a link or cut is toggled, a "changed" flag is set. This is used to signal the user that changes have been made if a quit command is executed.

2.1.3.3 Zooming

The DoZoom function is a loop which branches to the appropriate function based on the next event. It is shown in Figure 2.13. The only way to leave this loop is to either QUIT or change to EDIT. The function unique to this loop is that of zooming, which was explained earlier.

Zooming is implemented by doing a transformation of all points. The screen becomes a relative coordinate system. All input and output to the screen must be transformed into this relative coordinate system. This is done by piping all output to the screen through an output transformation and piping all mouse inputs through an input transformation. Each transformation does both a scale and a shift, as illustrated in Figure 2.14. Throughout the entire program, all input and output to and from the screen is run through the transformation functions.

Executing the zoom function simply involves changing the center point and the zoom factor in the transformation function and then redrawing the circuit. When doing successive "zoom ins," the old values are stored. That way the "zoom out" can return to the previous transformation function.

```
FUNC DoEdit

   LOOP
      Sleep Until Event;
      SWITCH (event)
         CASE (Expose Event):
              DoExpose;
         CASE (Button Pressed in Quit Menu):
              DoQuit;
         CASE (Button Pressed in Print Menu):
              DoPrint;
         CASE (Button Pressed in WriteCif Menu):
              DoWriteCif;
         CASE (Button Pressed in SaveLinkCut Menu):
              DoWriteLinkCut;
         CASE (Button Pressed in SaveCelled Menu):
              DoSaveCelled;
         CASE (Button Pressed in Zoom Menu):
              ******************************
              * User is leaving edit mode   *
              ******************************
              Put event back in queue;
              exit;
         CASE (Button Pressed in Draw Menu):
              IF Event location is close enough
                 to a link or cut
              THEN
                 Toggle the link or cut;
                 Set the CHANGED flag;
   END LOOP

ENDFUNC
```

Figure 2.12. The DoEdit function

```
FUNC DoZoom
   LOOP
      Sleep Until Event;
      SWITCH (event)
         CASE (Expose Event):
              DoExpose;
         CASE (Button Pressed in Quit Menu):
              DoQuit;
         CASE (Button Pressed in Print Menu):
               DoPrint;
         CASE (Button Pressed in WriteCif Menu):
              DoWriteCif;
         CASE (Button Pressed in SaveLinkCut Menu):
              DoWriteLinkCut;
         CASE (Button Pressed in SaveCelled Menu):
              DoSaveCelled;
         CASE (Button Pressed in Edit Menu):
              ********************************
              * User is leaving zoom mode    *
              ********************************
              Put event back in queue;
              exit;
         CASE (Button Pressed in Draw Menu):
              Get new center point;
              Increment Scale Factor;
              Generate transformation vector;
              Redraw Circuit;
   END LOOP

ENDFUNC
```

Figure 2.13. The DoZoom function

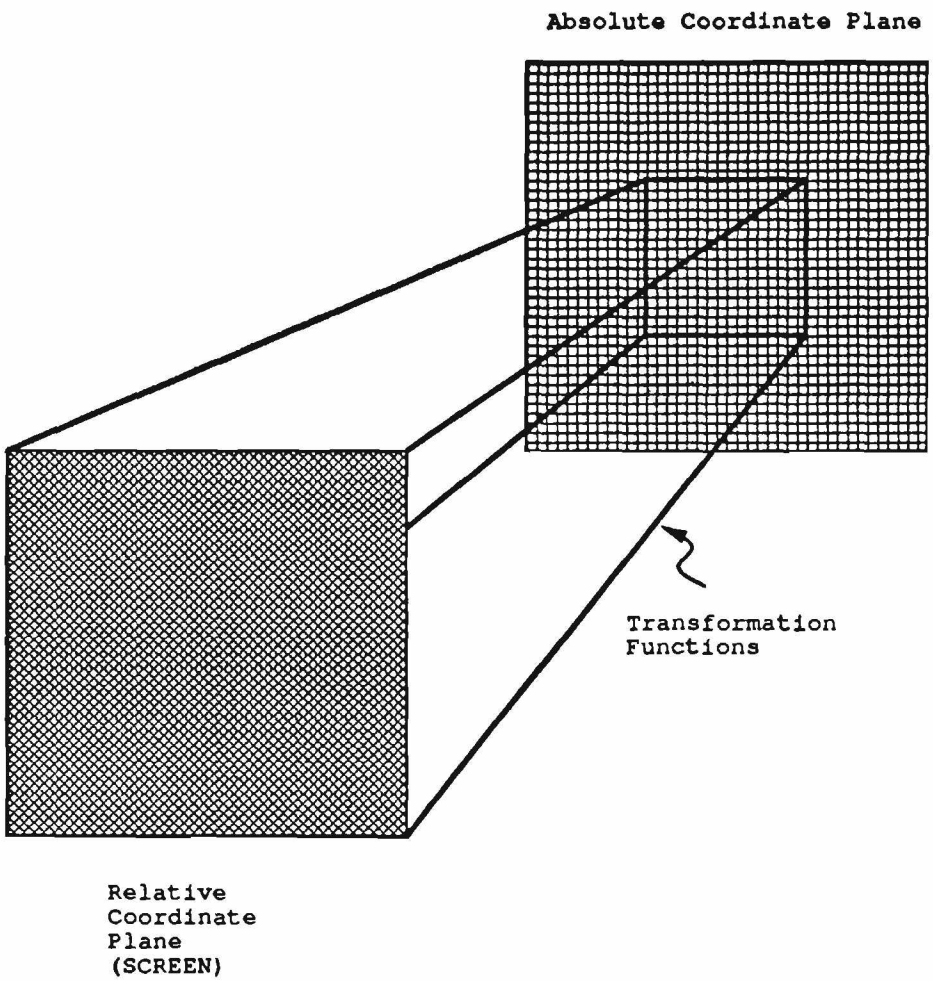


Figure 2.14. The transformation functions

2.1.3.4 Generating Files

The four file generation functions perform similarly. These include DoPrint, DoWriteCif, DoWriteLinkCut, and DoSaveCelled. All of them present a dialog box with which the name of the file can be modified or obtained. Once the file name has been changed, the modified version is the one presented in all subsequent dialog boxes. This incorporates the "Save As ..." command into the save command. Each type of file is saved as a file with a certain corresponding extension.

The DoPrint function is shown in Figure 2.15. This function opens a ".prt" file and write PostScript code to it. It scales the circuit so that it will fit completely on one page of output. The circuit looks almost identical to that displayed on the screen. The major difference is that the display uses diamond shaped filled polygons contacts instead of circles. This was done to speed up the redraw. Circles are very expensive computationally. On a print out, time is less important, so filled circles were used to represent contacts.

The DoWriteCif function is shown in Figure 2.16. It opens a ".cif" file and outputs CIF for all defined links and cuts. This CIF is then used for simulation as explained earlier.

The DoWriteLinkCut function is shown in Figure 2.17. It is very similar to the DoWriteCif function. It opens a ".cut" file and outputs the correct format for all defined links and cuts. The format is shown in Figure 2.4.

The DoSaveCelled function is shown in Figure 2.18. The output is written to a ".cel" file. This is the file which is read into CELLED. The value of each link and cut is simply a "1" or a "0" stored in the "cell" data structure.

2.1.3.5 Redrawing and Quitting

The DoExpose function is shown in Figure 2.19. It simply finds all windows that have been exposed and redraws them.

```
FUNC DoPrint
      Get file name through Dialog Box;
      Append a ".prt" extension to the file;
      Open the file;
      Print Header;
      Set Scale Factor;
      FOR each cell
      ****************************************
      * Print the base cell without the links *
      * and cuts.                             *
      ****************************************
          Print Core;

          For each cut/link
              IF TRUE
                 Print solid cut/link;
              ELSE
                 Print dashed cut/link;
      Print Tailer;
      Close File;

ENDFUNC
```

Figure 2.15. The DoPrint function

```
FUNC DoWriteCif

      Get file name through Dialog Box;
      Append on a ".cif" extension;
      Open the file;
      Print Header;
      FOR each cell
         IF the link/cut is TRUE
            Print the cif;
      Print Tailer;

ENDFUNC
```

Figure 2.16. The DoWriteCif function

```
FUNC DoWriteLinkCut

      Get file name through Dialog Box;
      Append on a ".cut" extension;
      Open the file;
      Print Header;
      FOR each cell
         IF the link/cut is TRUE
            Print the link/cut;
      Print Tailer;

ENDFUNC
```

Figure 2.17. The DoWriteLinkCut function

```
FUNC DoSaveCelled

      Get file name through Dialog Box;
      Append on a ".cel" extension;
      Open the file;
      Print Header;
      Print Size of Cell;
      FOR each cell
         For each link and cut
             Print the value
             of the link and cut;
      Print Tailer;

ENDFUNC
```

Figure 2.18. The DoSaveCelled function

```
FUNC DoExpose

   Find all windows exposed;
   Redraw them;

ENDFUNC
```

Figure 2.19. The DoExpose function

```
FUNC DoQuit

      Map The Quit Dialog Box;
      ********************************
      * Ask the user to confirm exit *
      ********************************
      LOOP
         IF confirmed
            Unmap Window;
            return with good exit condition;
         ELSE IF canceled
            return with bad exit condition;
         ELSE IF expose event
            DoExpose;
      END LOOP

ENDFUNC
```

Figure 2.20. The DoQuit function

The final function detailed is the DoQuit function shown in Figure 2.20. When quitting, a dialog box is always presented. If changes have been made, a warning reflecting this is given. If no changes have been made, the user still must confirm the exit. The user always has the option of cancelling the exit.

2.2 Existing Tools

With the completion of CELLED, the next step was to integrate the RIPPL cell set into the existing PPL tool suite. This would allow the designer the use of the PPL tools in a manner similar to the design with CMOS or GaAs technologies.

2.2.1 TILER

TILER is the tiling editor used with PPL. It represents cells with ASCII characters. This allows circuits to be designed without any graphical terminal display capabilities. TILER is technology independent. CMOS, NMOS, and GaAs cell

Figure 2.21. The display of a JK flip-flop in TILER

sets have all been built and used successfully. An example of how TILER would display a JK flip-flop is shown in Figure 2.21. It is drawn much larger than actual scale. Everything is an ASCII character. The quotation marks are fill cells. This means that the JK flip-flop occupies a 2 x 3 area. The vertical lines on the top row represent cuts. For further details see [1].

TILER uses a specific data base format to display the cells. This data base is stored in a TCH file, standing for TeCHnology file. Each technology must have its own technology file. This technology file contains information such as the size of a cell, any forced breaks, and the ASCII letter used to represent to cell.

Two new technologies were developed for RIPPL. They are called LASERT and LASERNT, meaning LASER Testable and LASER NonTestable. To the designer, they both resemble the CMOS technology. Most cells have the same look and feel as CMOS. When the user starts TILER, either "LASERT" or "LASERNT" is typed in as the technology.

Complete technology files were developed for the LASERT and LASERNT technologies. Presently they each contain approximately 30 cells. These include routing cells, pad cells, inverters, n-input NAND cells, multiplexers, 2 input NAND and NOR cells, and D and JK flip-flops.

Every time a designer wishes to add a cell to the cell set, the technology files

need to be modified. The technology file is easy to understand and manipulate. Modification should not present a challenge to the designer.

2.2.2 SIMPPLEX and SIMPPL

SIMPPLEX is the circuit extractor and SIMPPL is the circuit simulator for PPL. They both are technology independent, and both use the same data base file, namely the Simulator Data Base (SDB) file.

The SDB file contains a description of a PPL cell set at the logical or functional level. PPL cells are described in terms of primitives that are understood by the simulator.

An SDB file must be created for every technology created. Two new simulator data base files were created for the LASERT and LASERNT technologies.

Every time a new cell is added to the cell set, the SDB files must be changed. These files are much more difficult to understand and develop than the TCH files. It is hoped that a future addition to the PPL software will be a compiler for both SDB and TCH files. This would greatly facilitate the generation and modification of these files.

2.2.3 PPL2CIF

PPL2CIF is the program used to generate CIF files. RIPPL circuits do not need a CIF file for fabrication. The circuits have already been prefabricated. All that is needed is a link/cut list, but there is still a use for the PPL2CIF tool. It is always safest to extract and simulate the file before laser zapping the circuit. Since SIMPPL uses a logical description of the circuit to simulate, there is a possibility that the description was incorrect. Simulating the CIF allows an extra assurance that the SIMPPL simulation was correct, so the use of PPL2CIF is strongly recommended.

```
DS BLANK 100 1;
9 BLANK;
C POWTAP T 0,0;
L CMF;
P -12,-10 -12,9 -9,9 -9,-10;
DF;
DS POWTAP 100 1;
9 POWTAP;
C VDDTAP T 0,0;
C GNDTAP T 0,0;
DF;
DS VDDTAP 100 1;
9 VDDTAP;
L CWN;
P -25,-11 -25,11 0,11 0,-11;
L CAA;
P -25,6 -25,11 -23,11 -23,6;
B 2 4 -24,-9;
DF;
DS GNDTAP 100 1;
9 GNDTAP;
L CWP;
P 0,-11 0,11 25,11 25,-11;
L CAA;
P 23,6 23,11 25,11 25,6;
B 2 4 24,-9;
DF;
```

Figure 2.22. An example of SIF

The PPL2CIF program uses a modified version of CIF named SIF (pSuedo cIF). It is very much like CIF. An example of SIF is shown in Figure 2.22. This is not the actual SIF for a circuit but is used only as an example of a SIF file. Some of the differences between CIF and SIF include using names instead of numbers and reversing the order of hierarchy calls. A SIF library can be easily built using the CIF files generated by CELLED.

Complete SIF libraries have been generated for the LASERT and LASERNT technologies. A file generated with PPL2CIF can be simulated using the VTI tools.

When using the MOSIS Tiny Chip pad frame, names are automatically placed on the pads. This facilitates the simulation.

2.3 MakeZap

Originally it was hoped that the PPL2CIF program could be modified to generate a link/cut list. This was unsuccessful. The problem encountered was due to the difference in the way wire breaks are defined in PPL and RIPPL. In PPL, a wire break is made by not calling a connection i.e., by not doing anything. In RIPPL, a wire break is made by calling a cut location in the link/cut list i.e., by doing something. This incompatibility could only be eliminated through large program changes to the PPL software and was not done.

In order to get circumvent this problem, a program called MakeZap was written. It is a C shell script which uses the UNIX utilities AWK and SED. It converts a flat CIF file into a "ZAP" (link/cut list) file. The CIF file used in this program is not the CIF file generated by PPL2CIF. The CIF file must be modified in two ways. First, there must exist a way of distinguishing between the different types of cuts. Second, the file must be flat i.e., it cannot contain any hierarchy. Normally, the PPL2CIF file uses only the "exclude" layer to define a cut. This works for all types of cuts. In order to be able to distinguish between the types of cuts, the model for a cut was changed to include both a box of "exclude" and a box of the layer being cut. The boxes are the same size and are placed on top of each other. This allows the CIF to be extracted and simulated correctly since the "exclude" layer tells the extractor to ignore any geometry below it. At the same time, the type of cut can be distinguish by the type of box underneath it. This solved the first problem. The second problem was solved using the VTI tools, which have the ability to flatten a CIF file. Both the LASERT and LASERNT CIF libraries were modified to include the layer underneath the "exclude" layer.

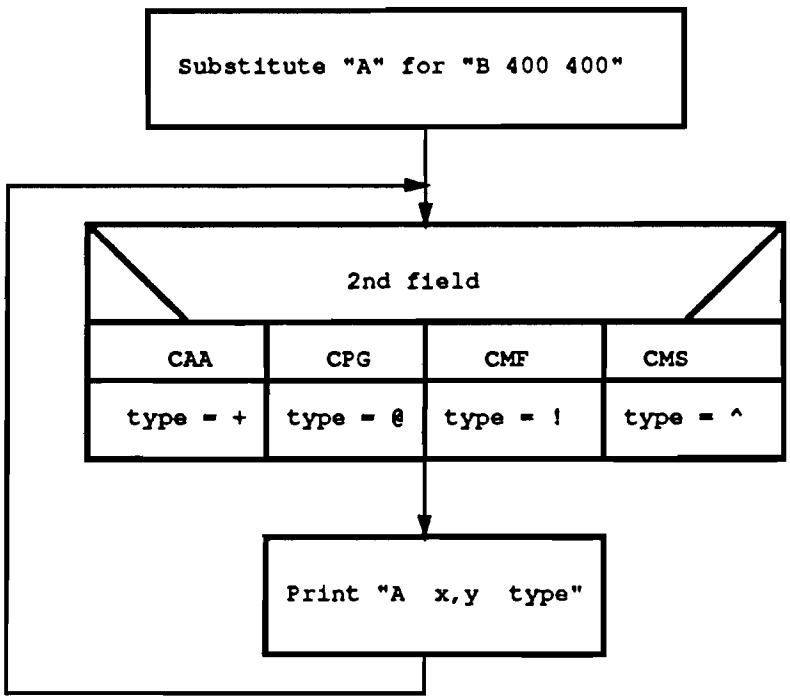


Figure 2.23. The basic flow chart for the MakeZap program

To use MakeZap, the user types "makezap FileName". The FileName argument is a CIF file without the ".cif" extension. The zap file is written to FileName.zap.

The basic flow chart for the MakeZap program is shown in Figure 2.23. This includes both the SED and AWK script.

CHAPTER 3

RIPPL CELLS

3.1 Nontestable cells

Included in this section are the schematics and link/cut diagrams for the cells employed in the current version of the RIPPL cell set.

3.1.1 Basic Cells

Figure 3.1, Figure 3.2 and Figure 3.3 are the basic cells.

3.1.2 N-input NAND Cells

Figure 3.4, Figure 3.5, Figure 3.6 and Figure 3.7 are the N-input NAND cells.

3.1.3 Inverter Cells

Figure 3.8, Figure 3.9, Figure 3.10 and Figure 3.11 are the inverter cells.

3.1.4 Flip-flop Cells

Figure 3.12 and Figure 3.13 are the flip-flop cells.

3.1.5 Dynamic Latch Cells

Figure 3.14 and Figure 3.15 are the dynamic latch cells.

3.1.6 Multiplexor Cells

Figure 3.16 and Figure 3.17 are the multiplexor cells.

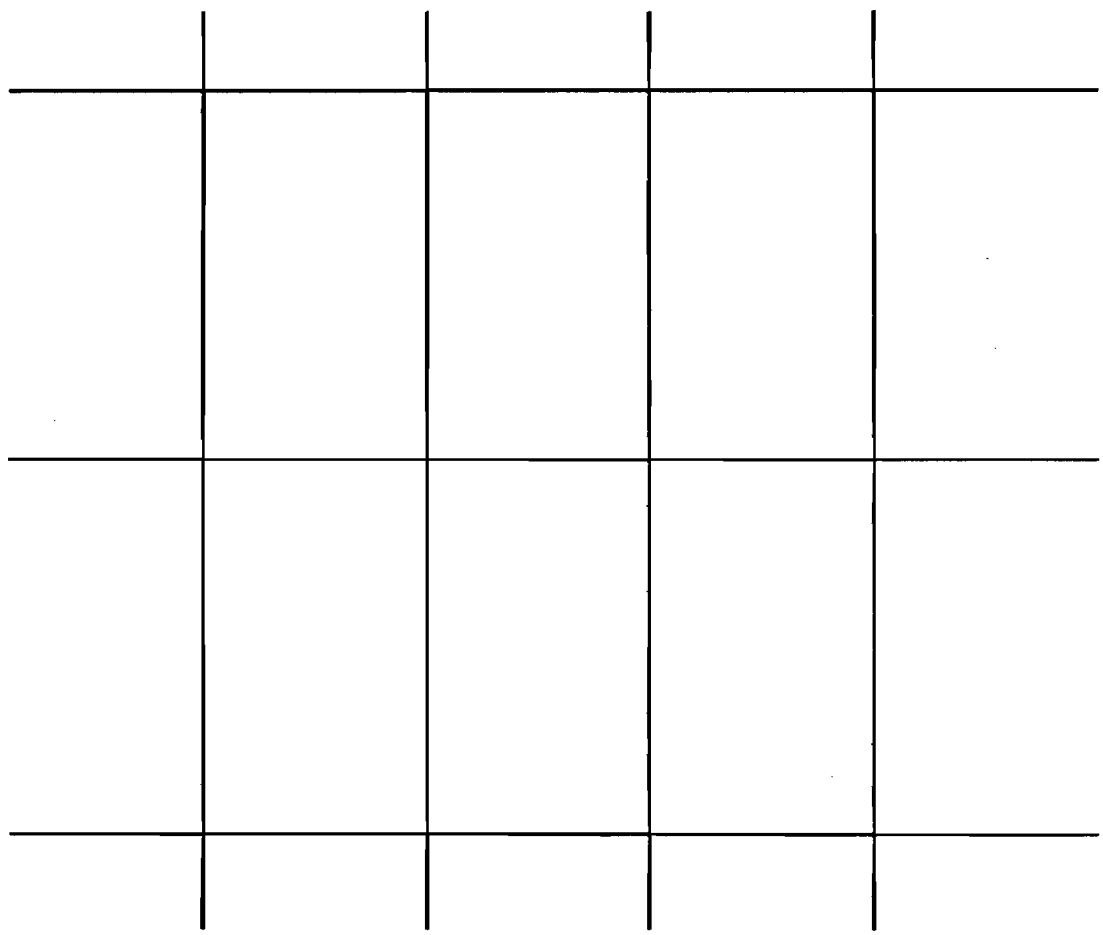
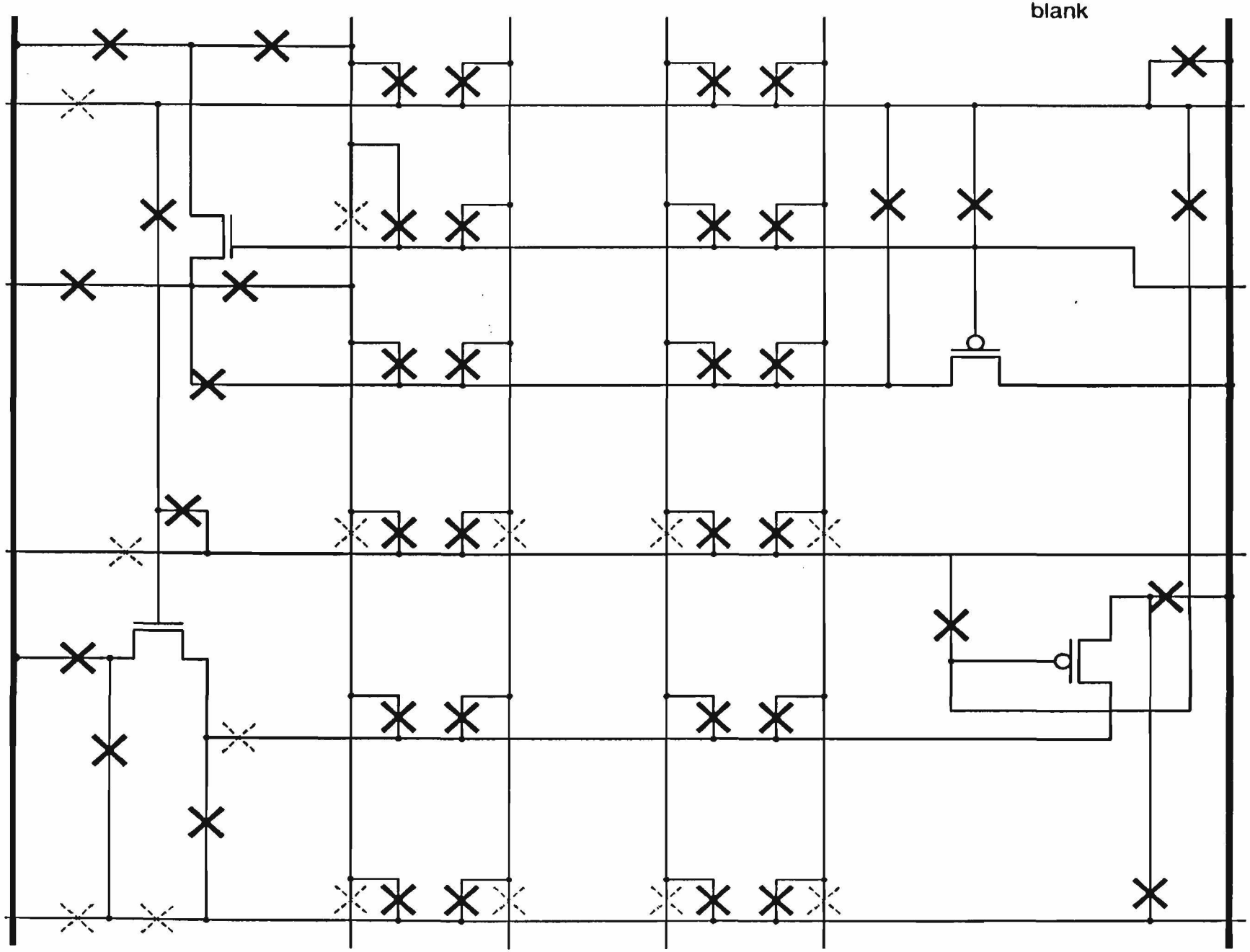


Figure 3.1. The BLANK cell

blank



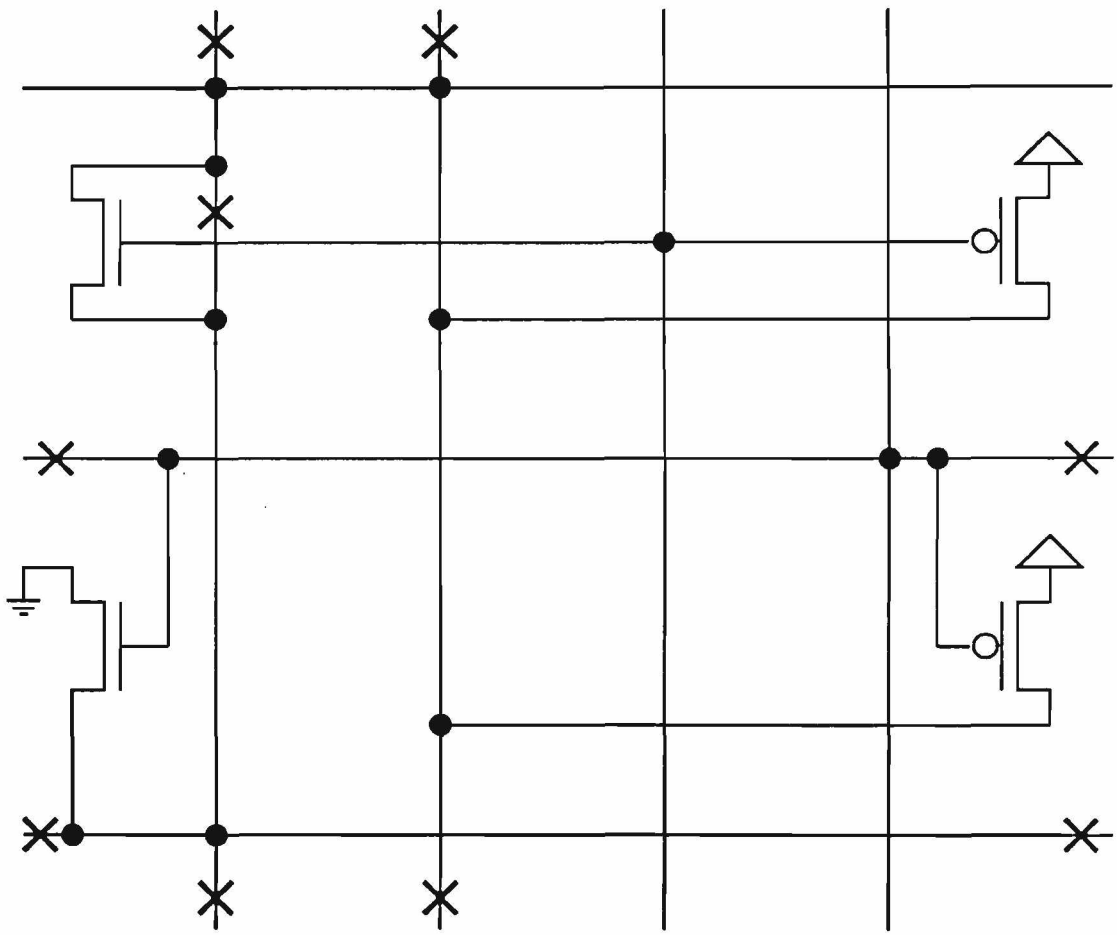
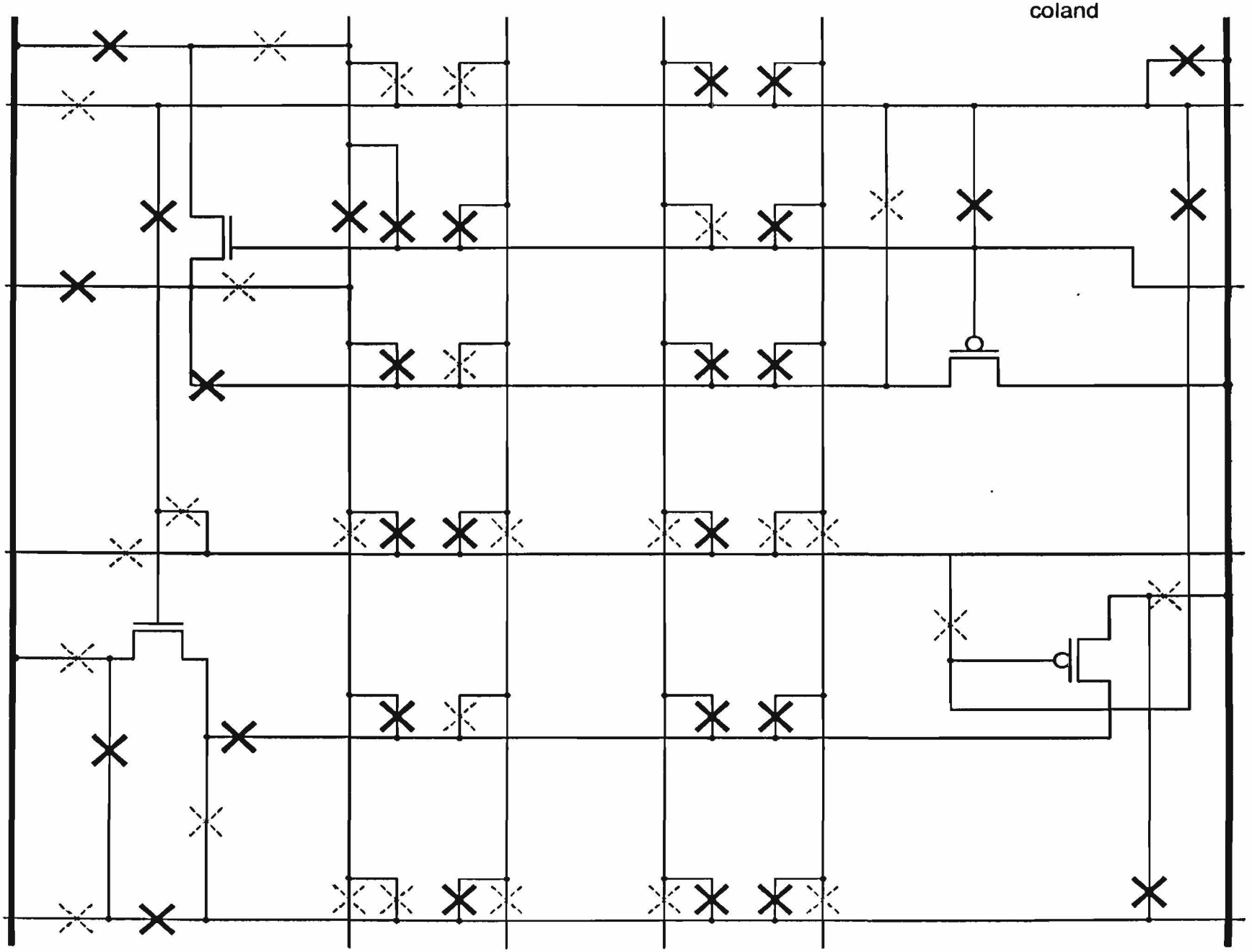


Figure 3.2. The NAND cell



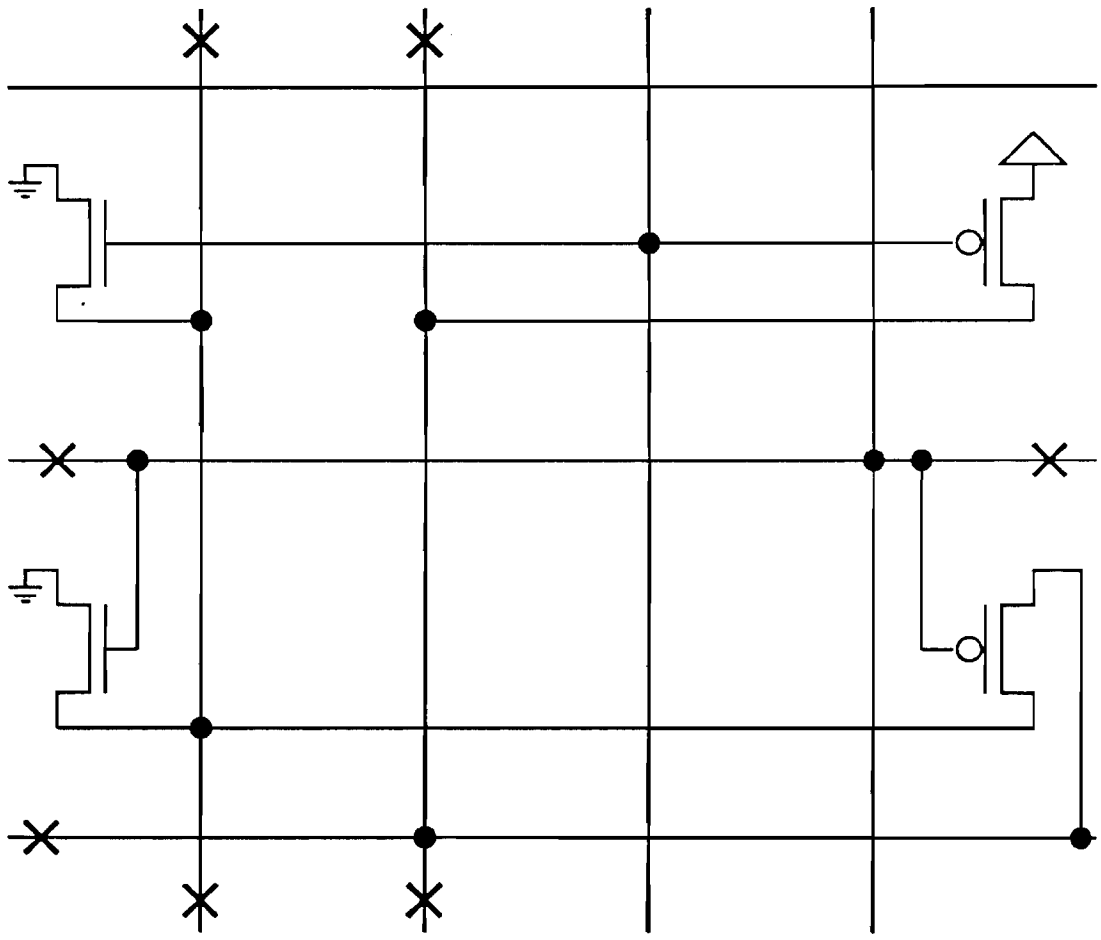
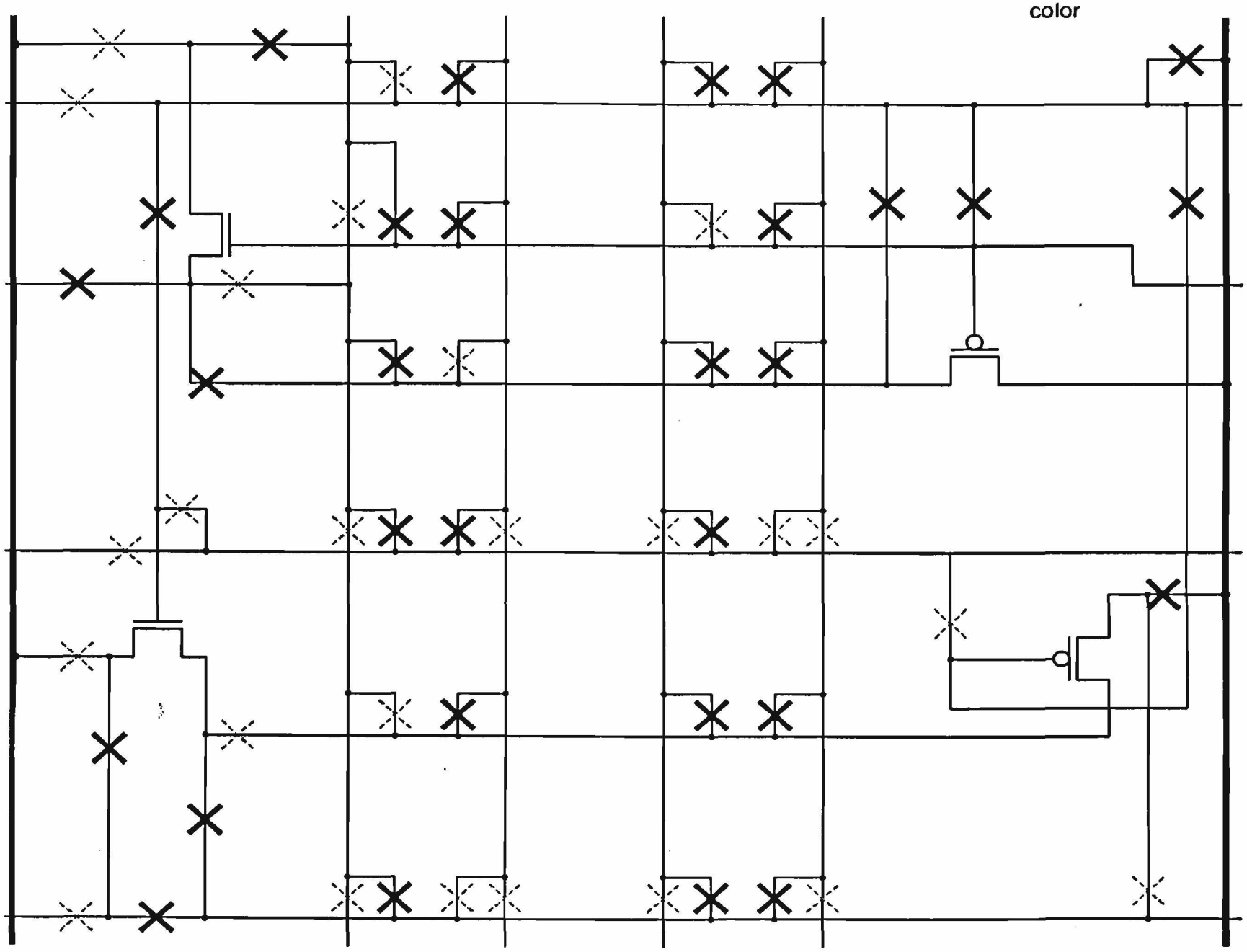


Figure 3.3. The NOR cell


color

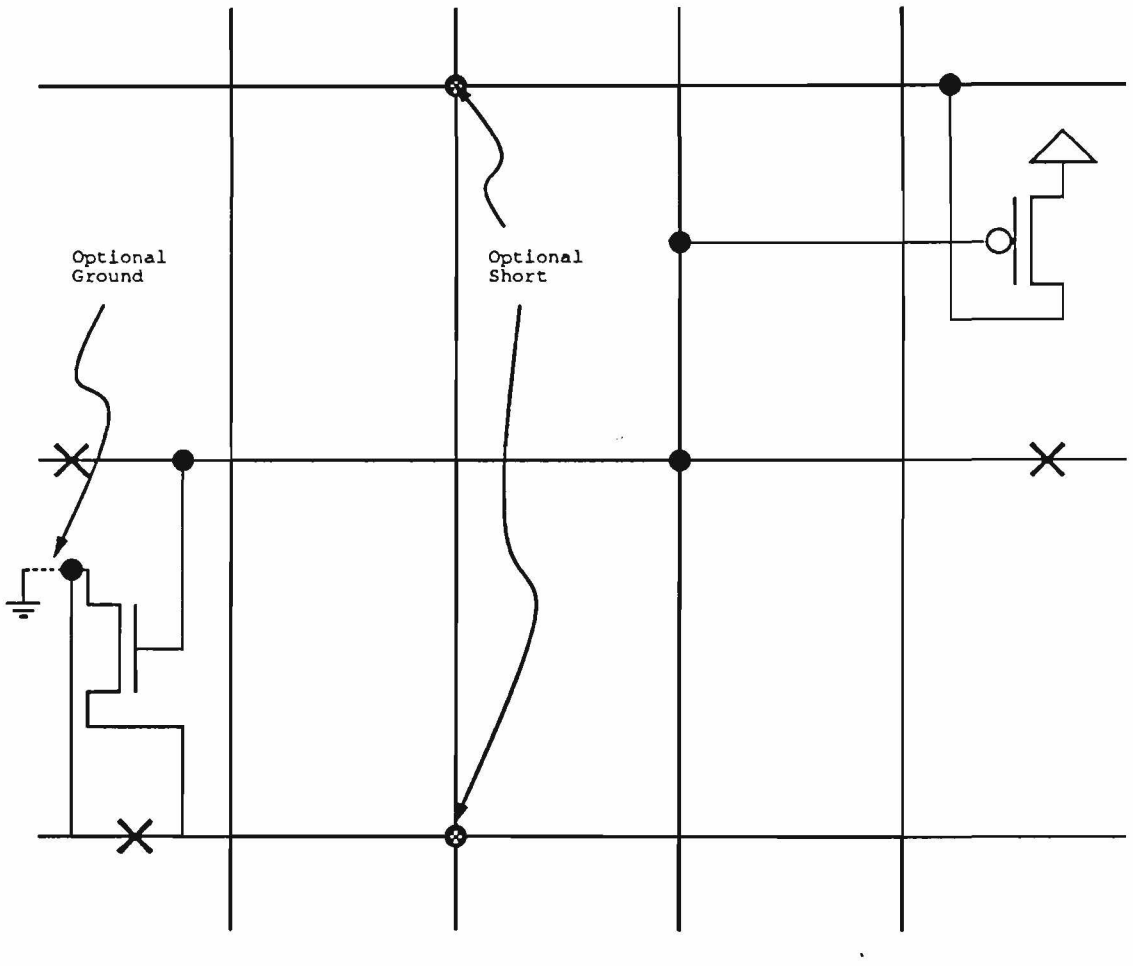
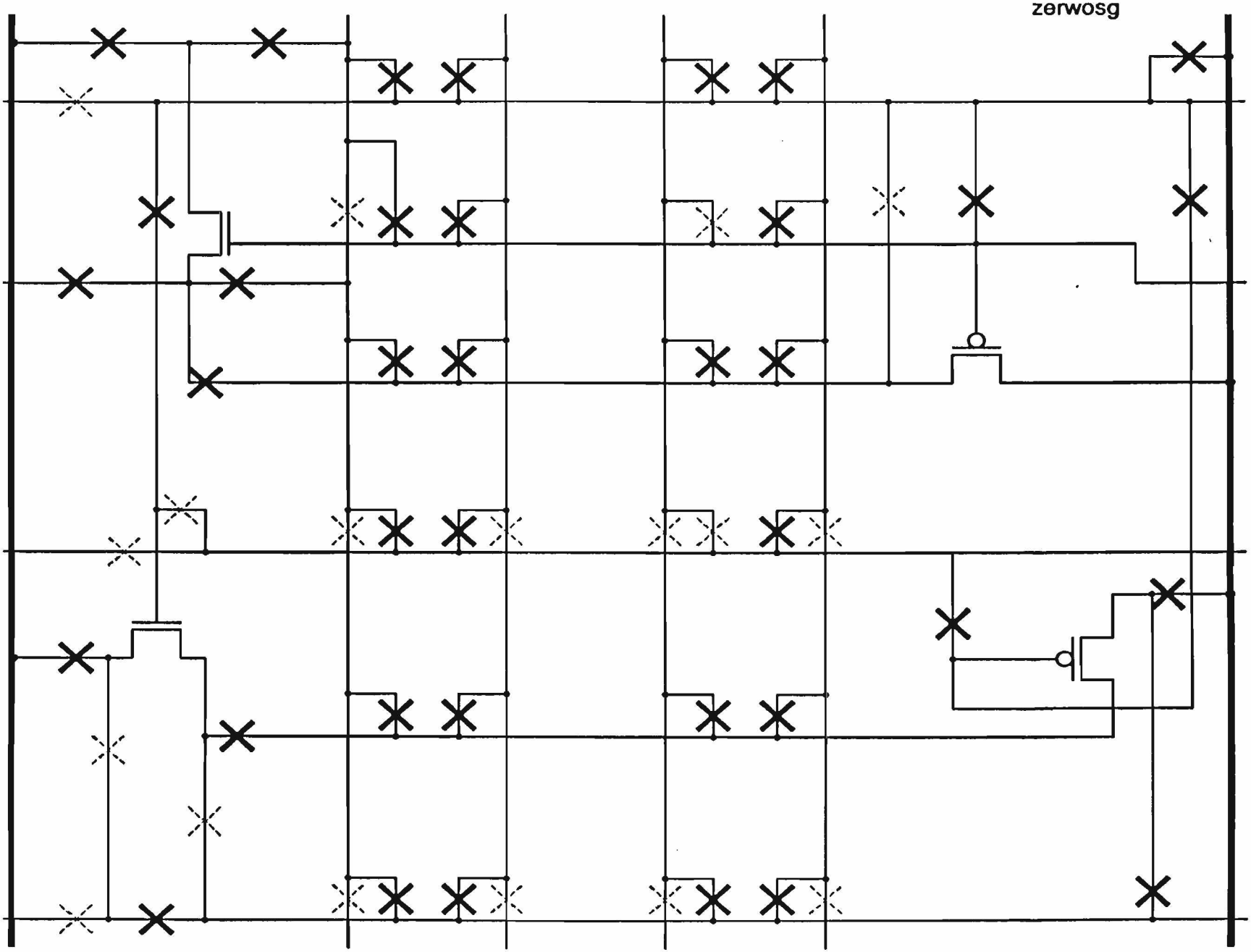
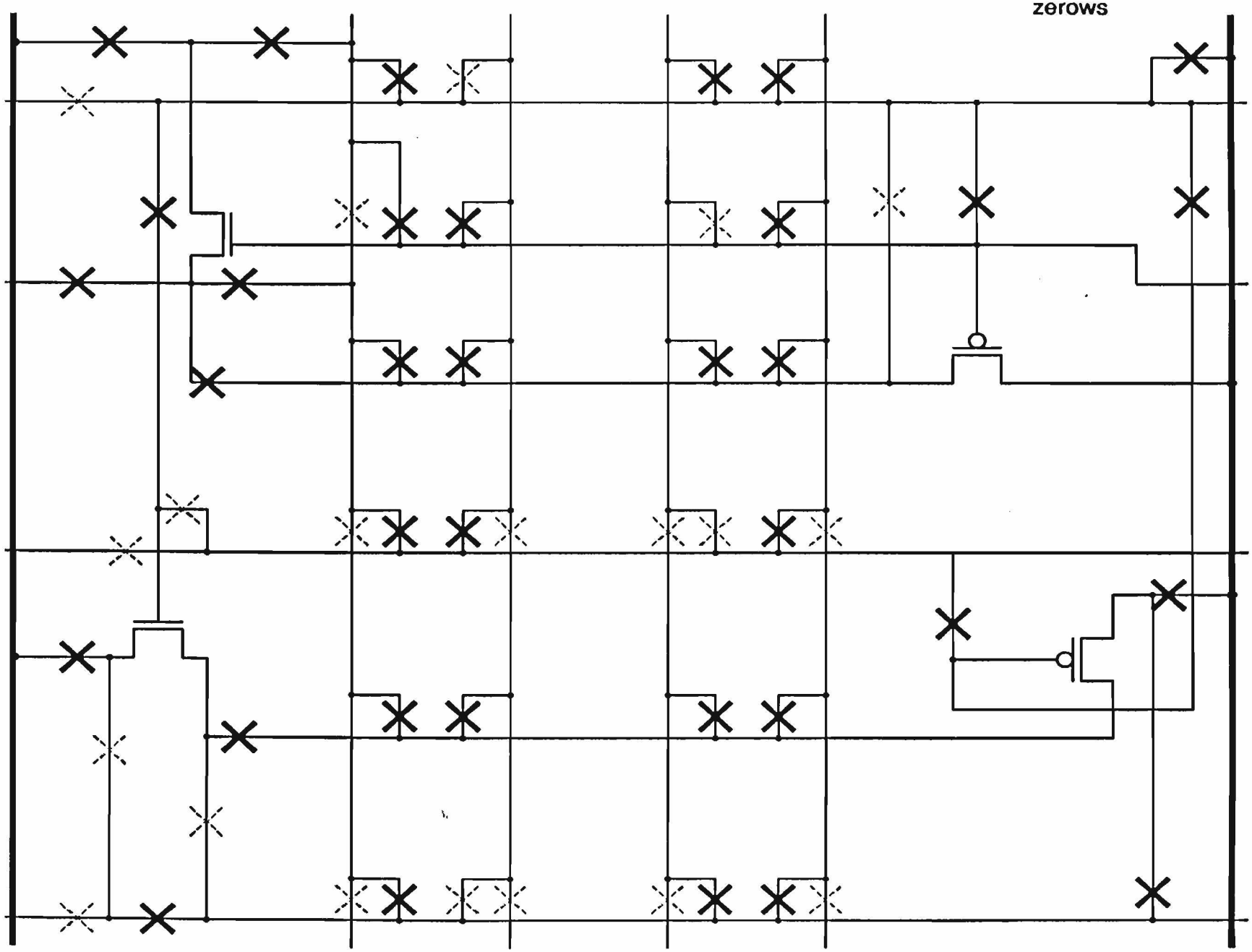
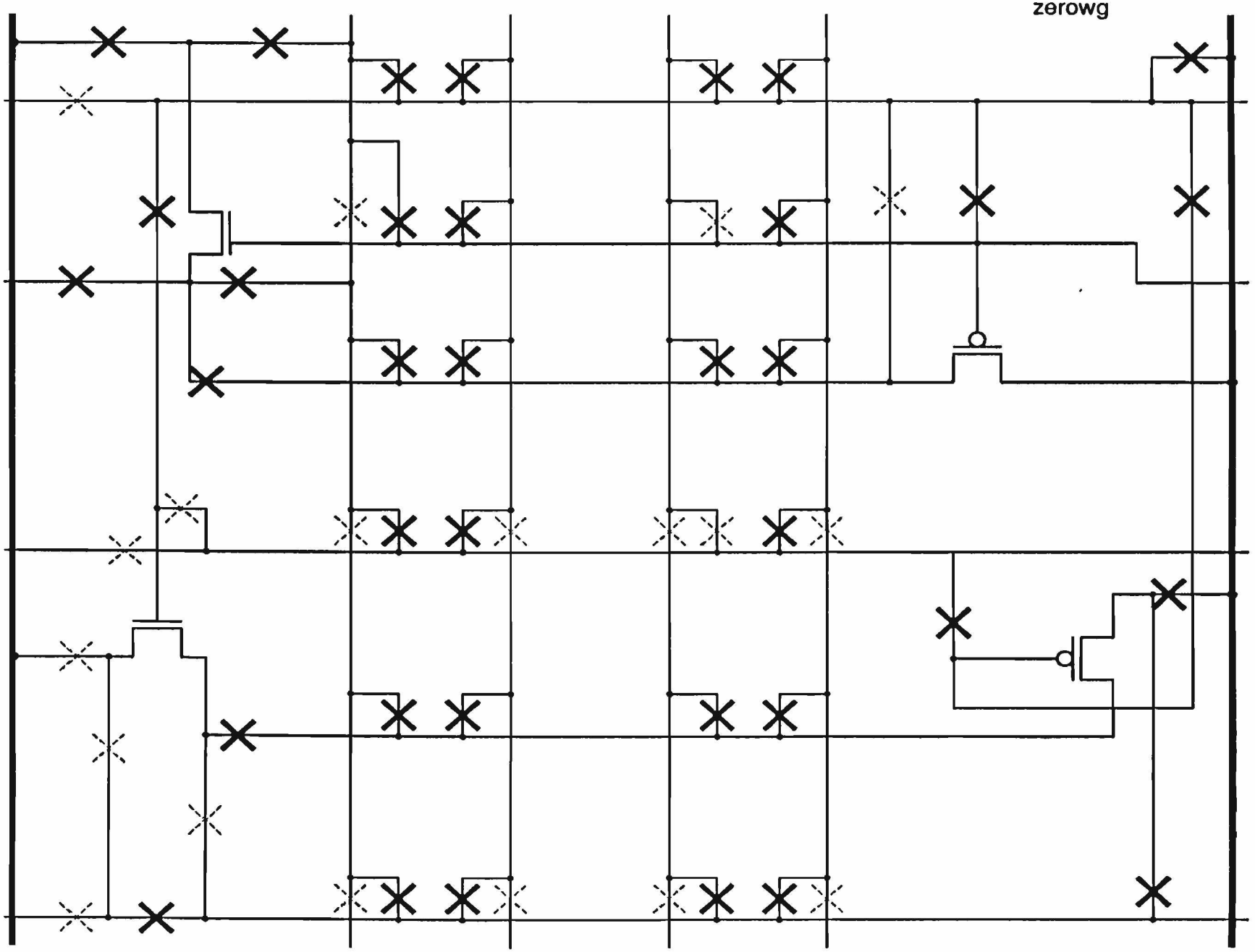
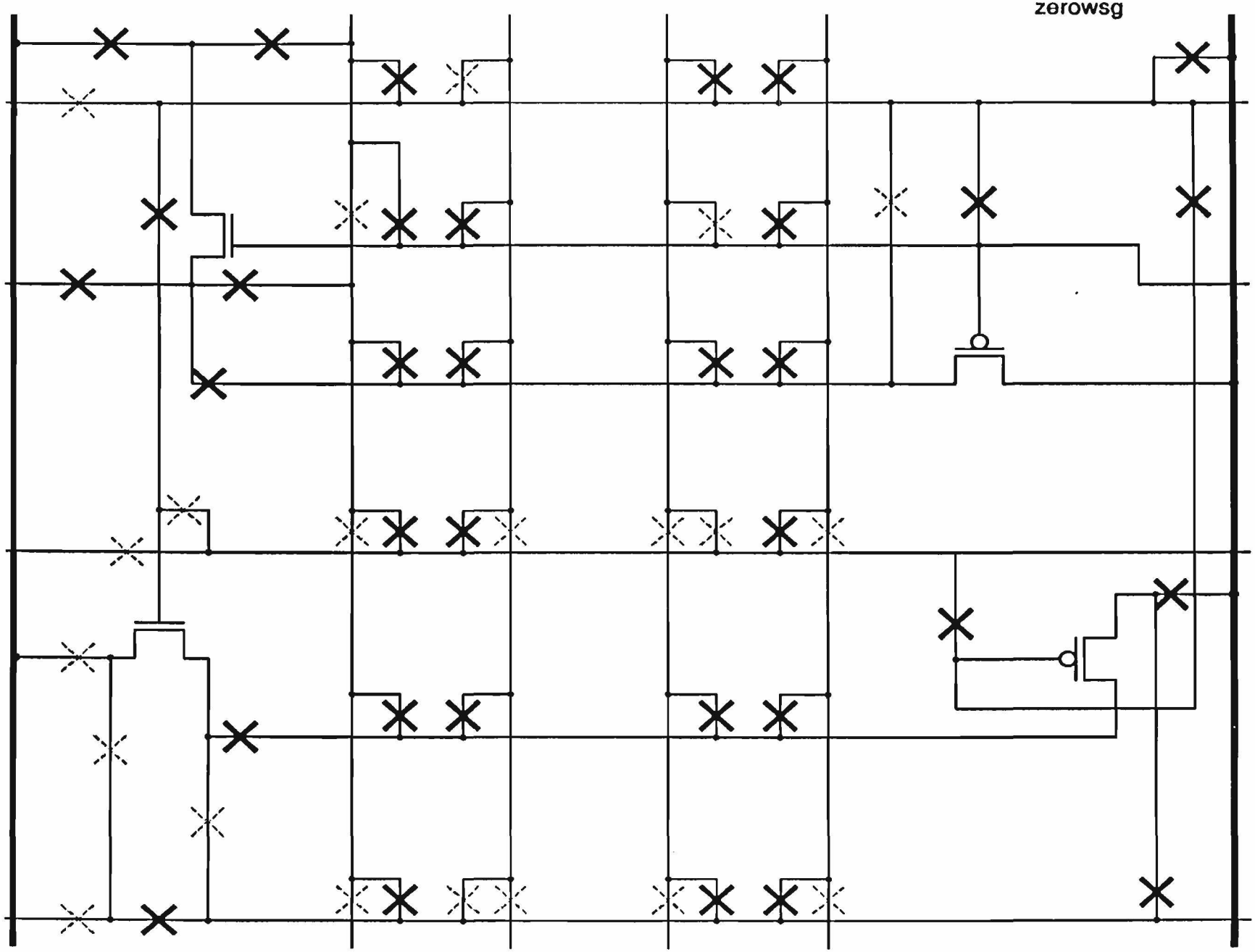


Figure 3.4. The 0 cell


zerwosg


zerows


zerowg


zerowsg

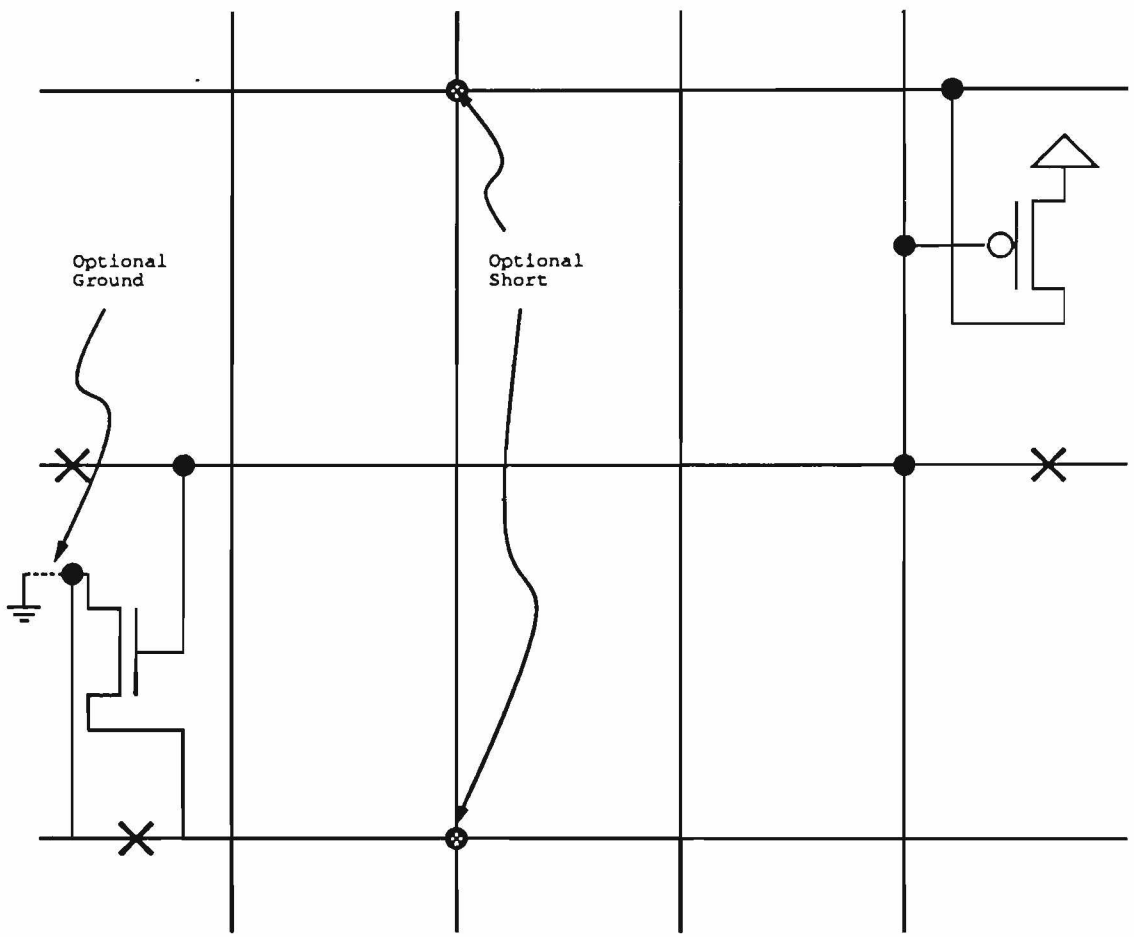
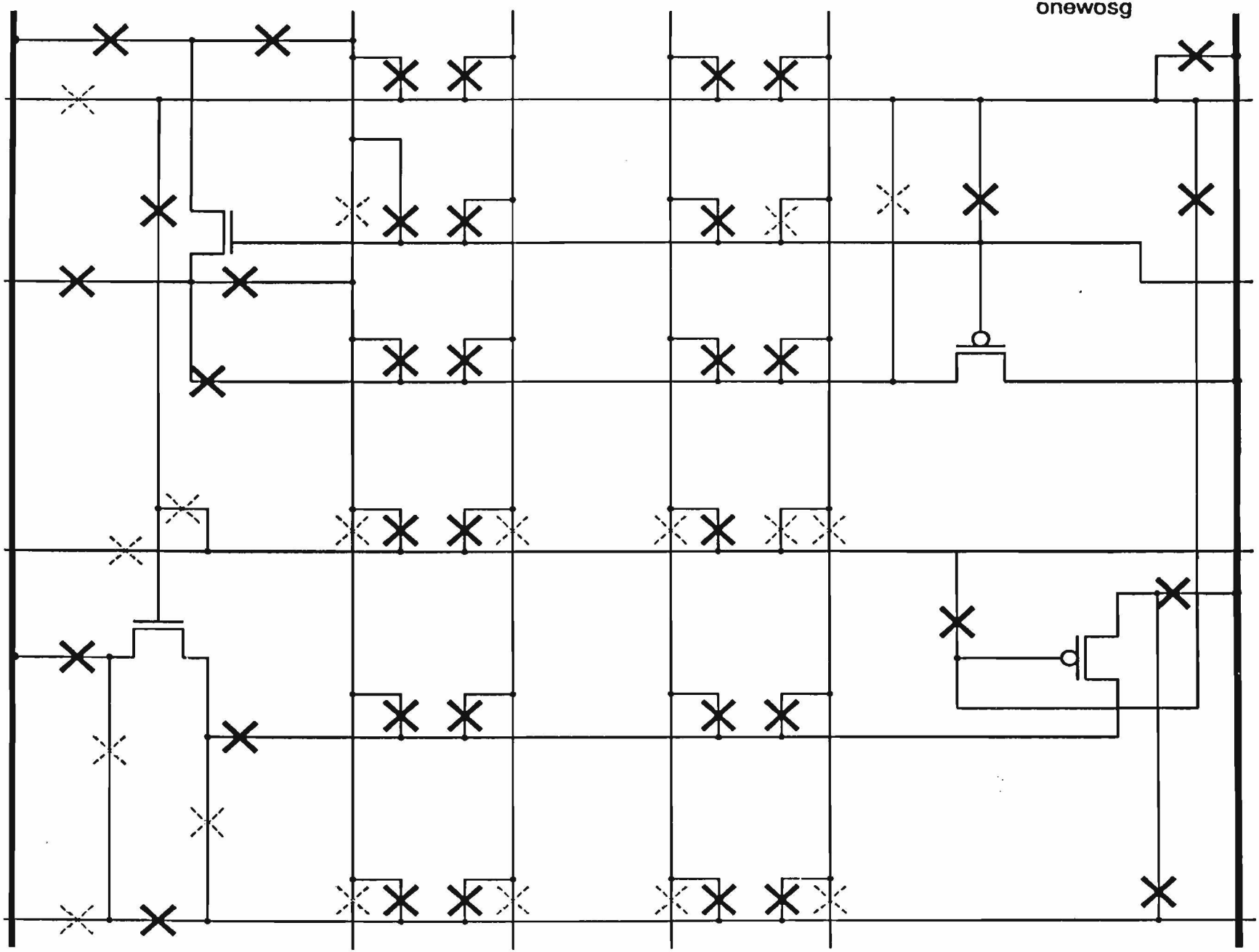
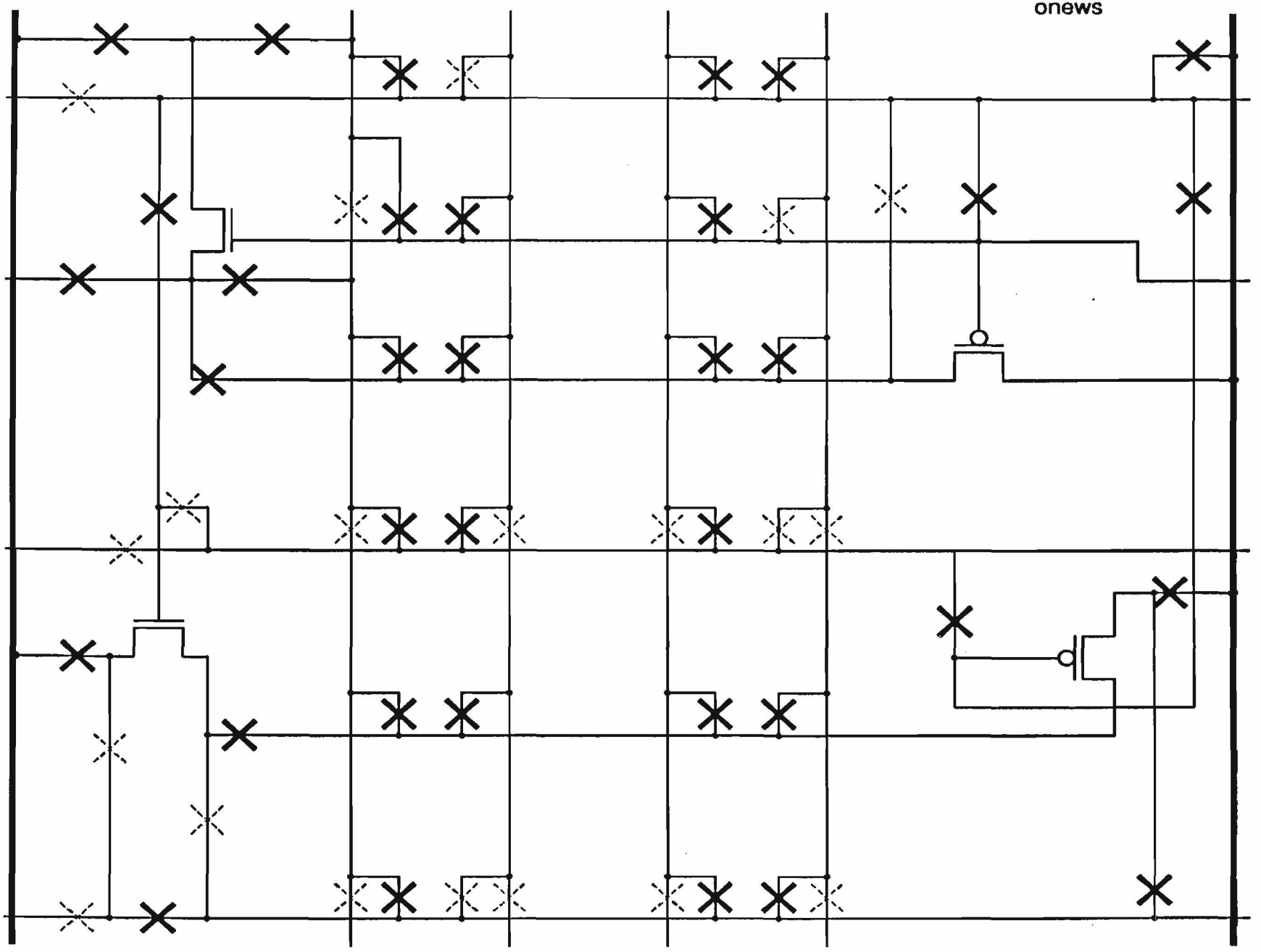
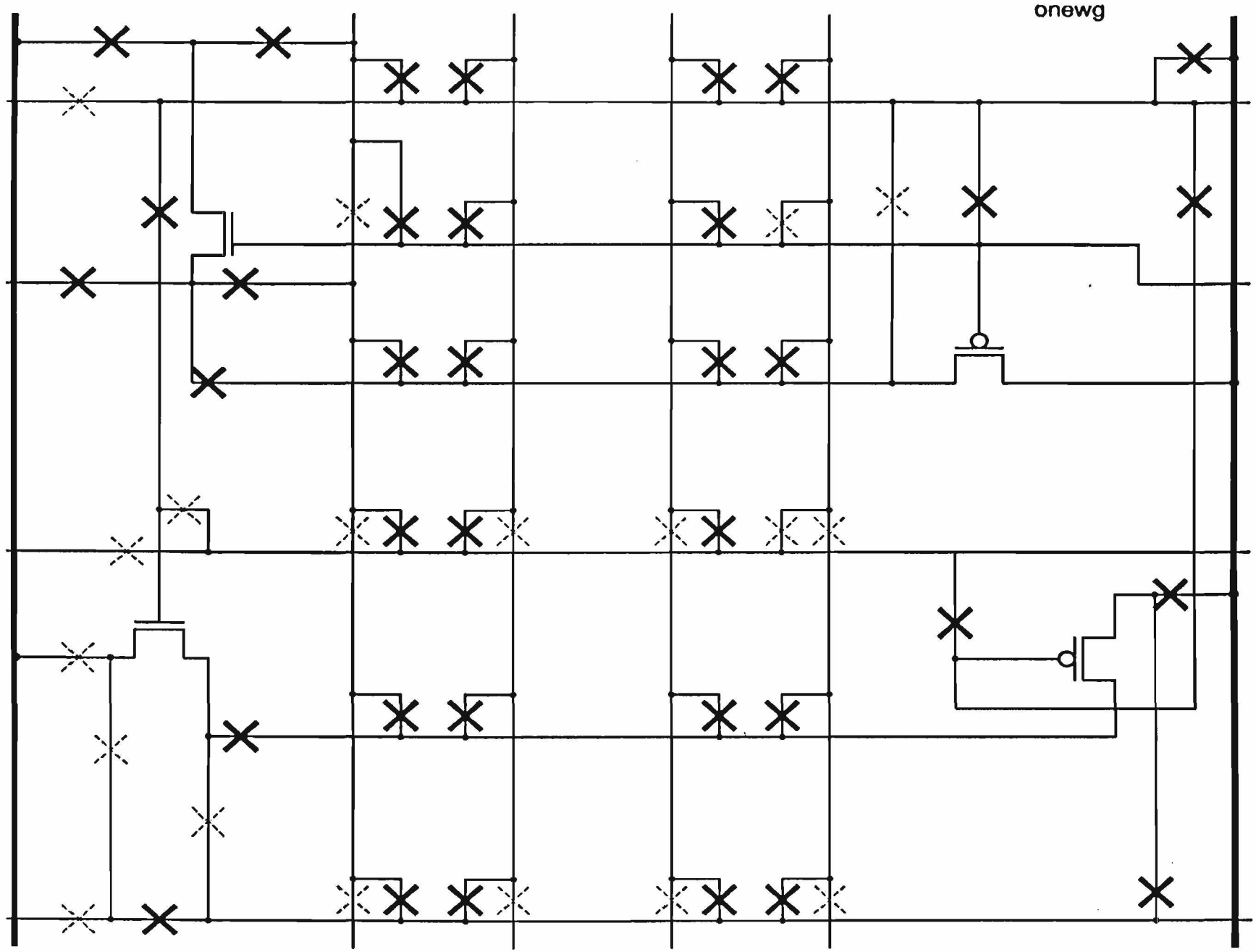
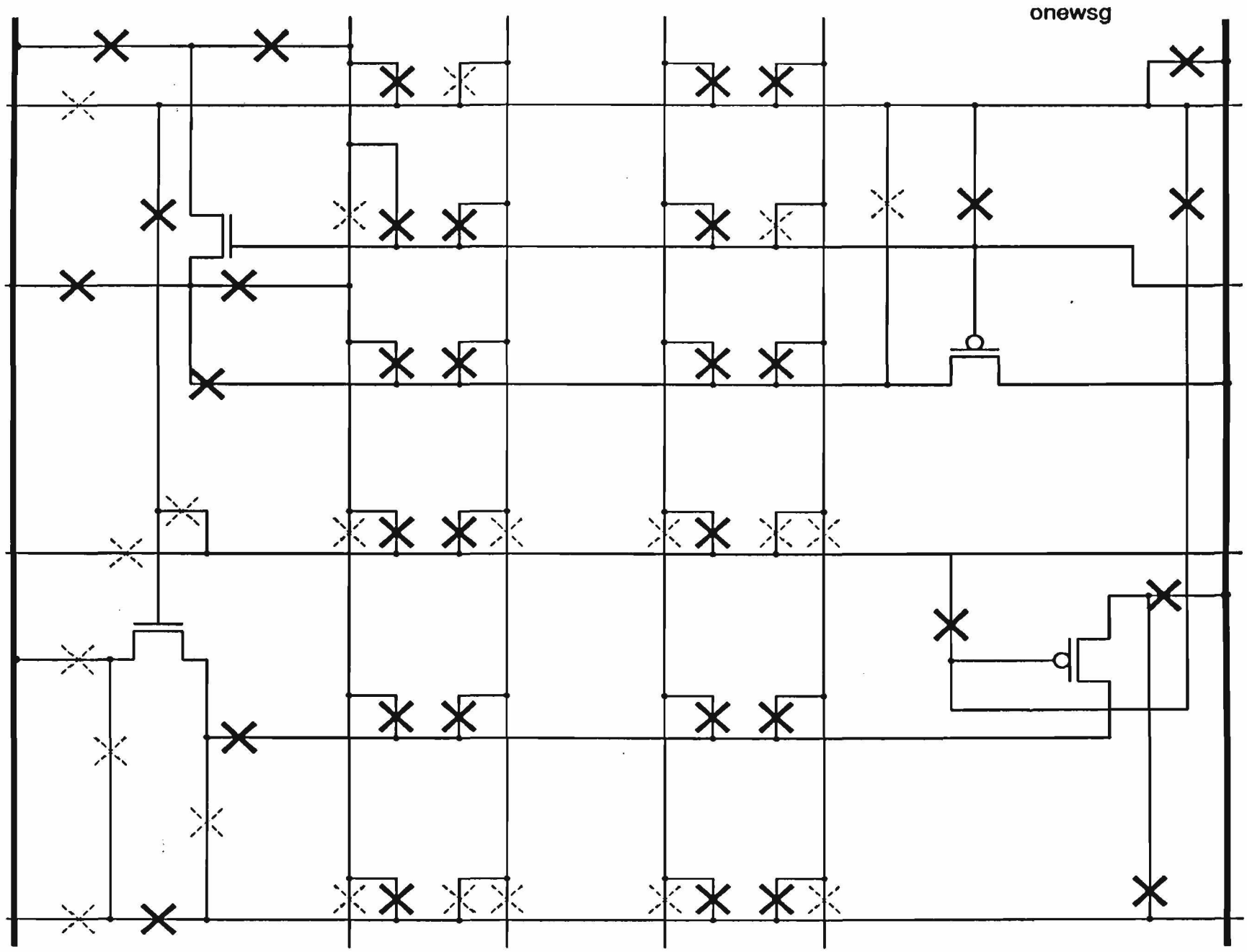


Figure 3.5. The 1 cell


onewosg




onewg


onewsg

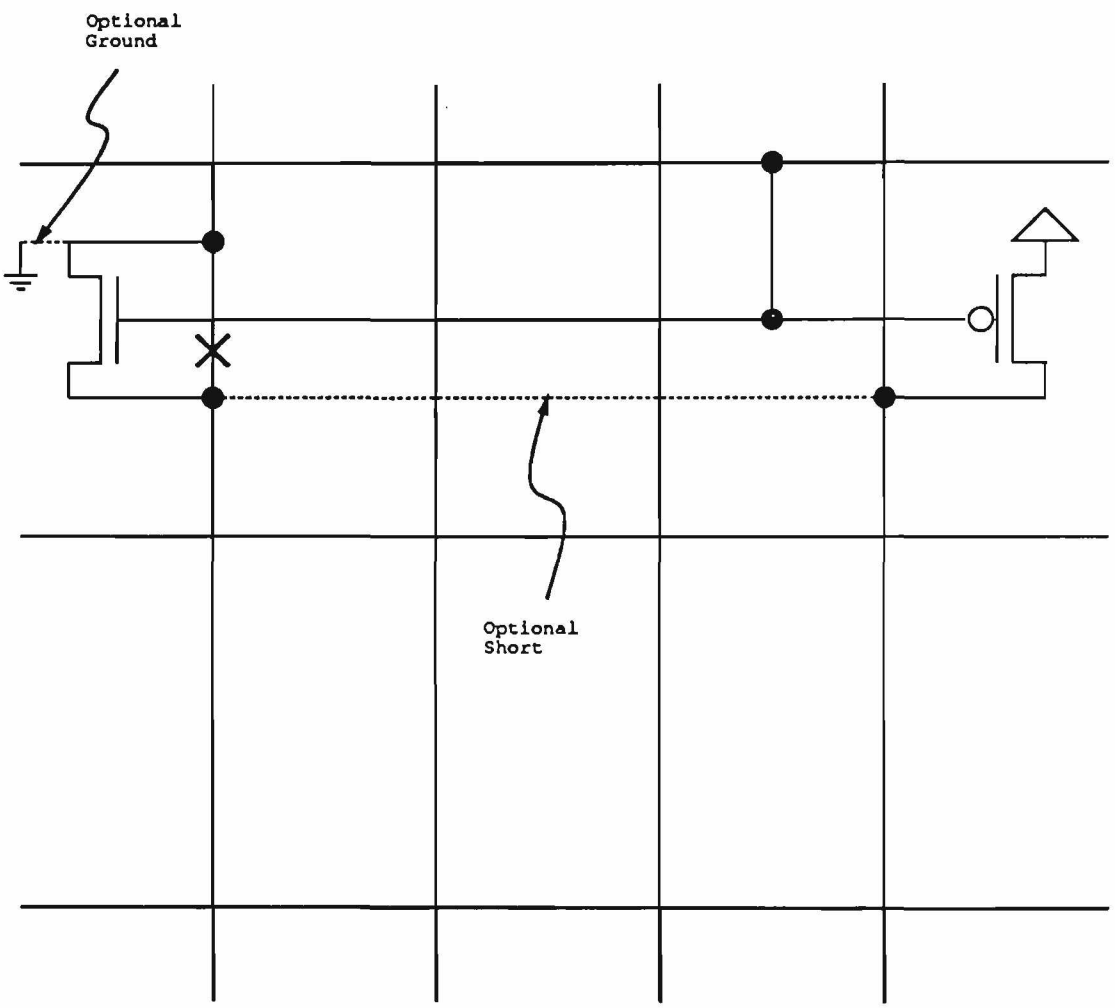
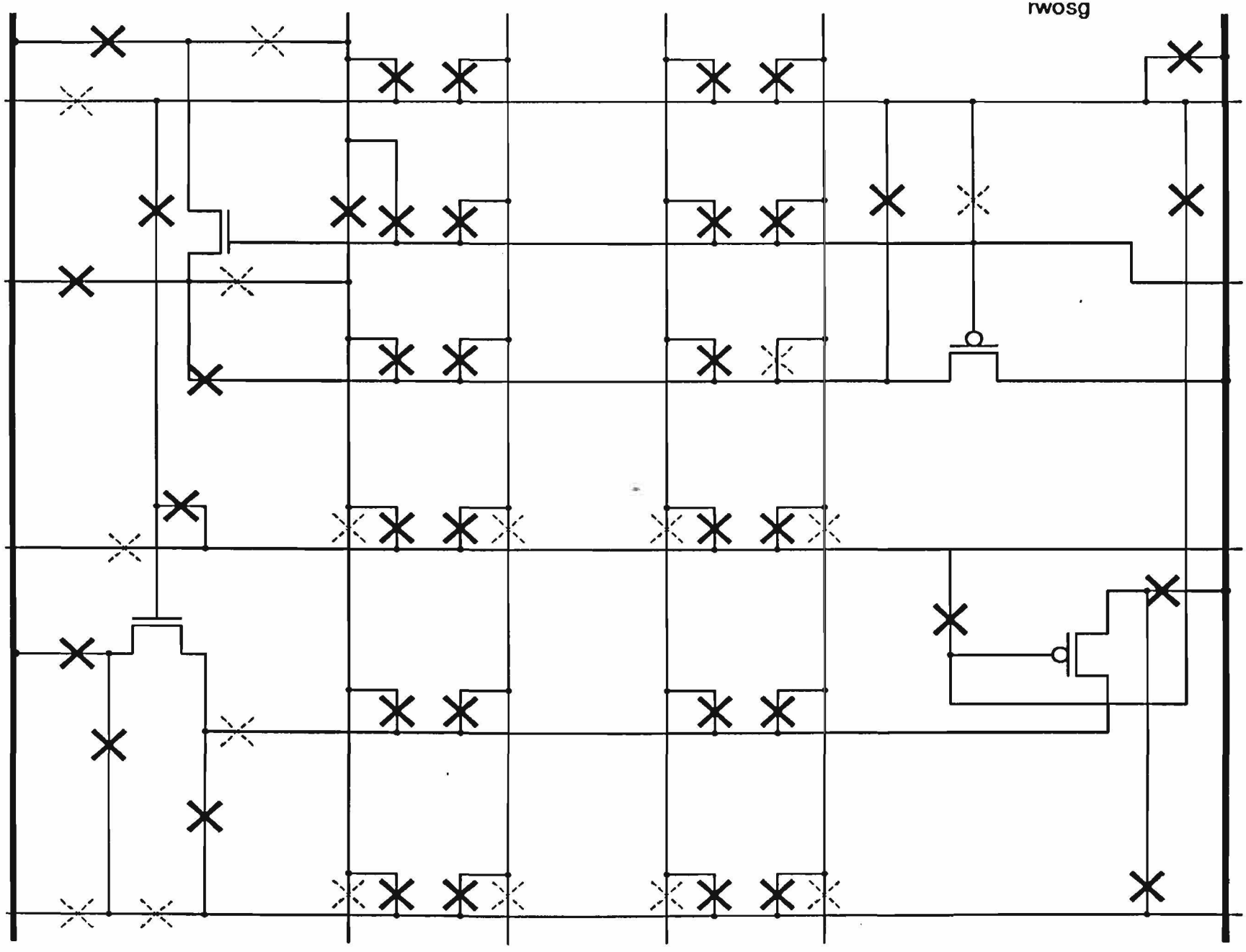
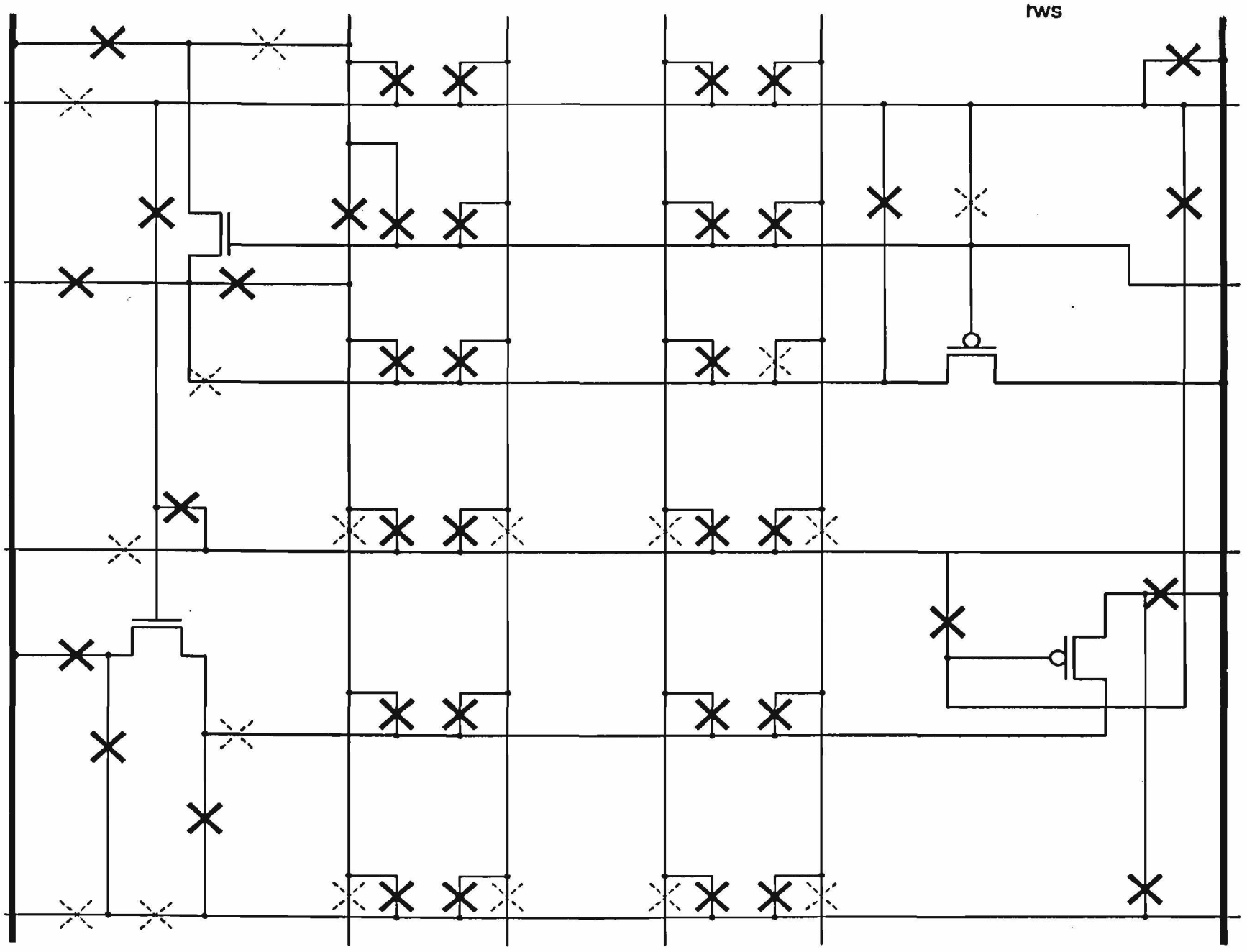
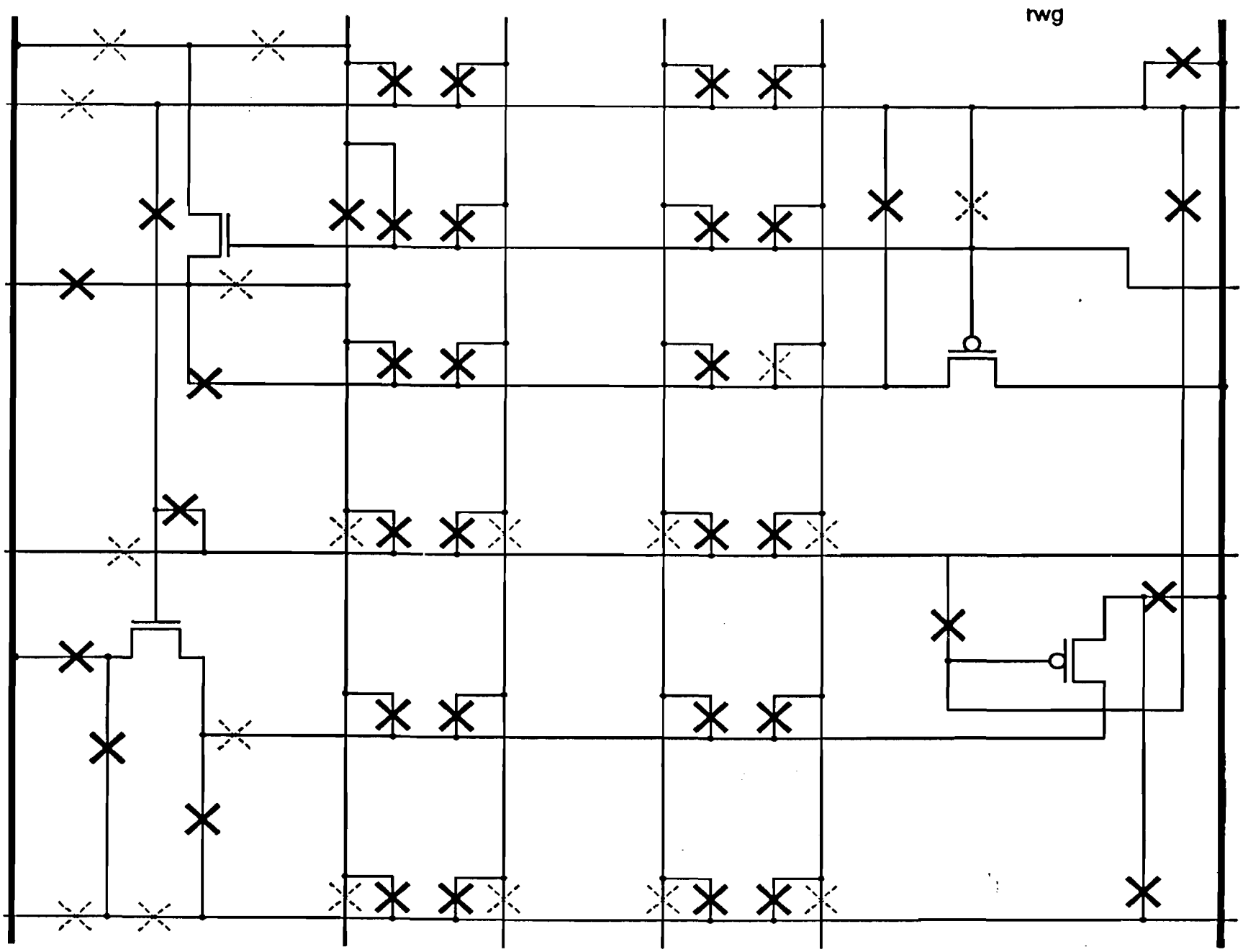
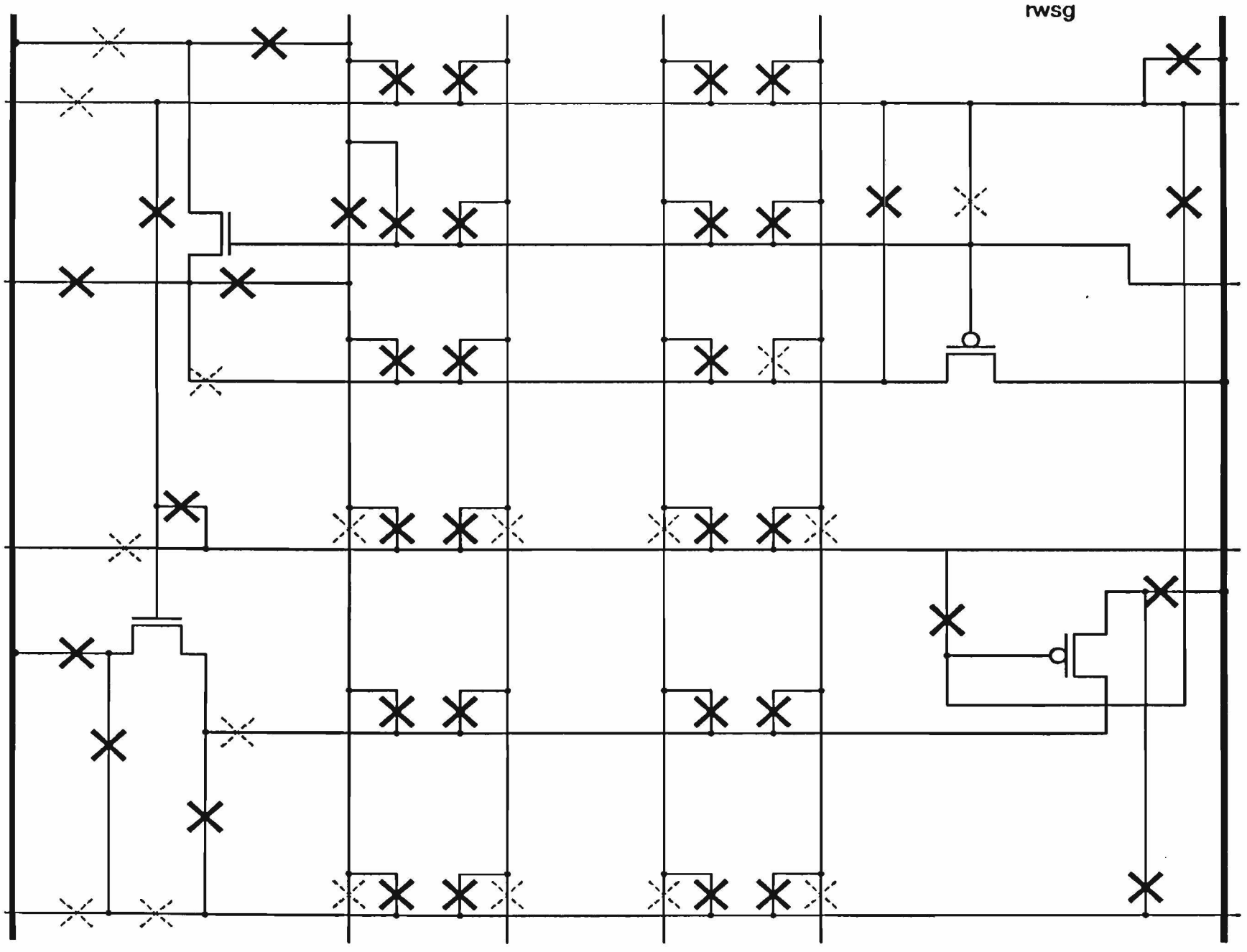


Figure 3.6. The r cell


rwosg


rws


rwg


rwsg

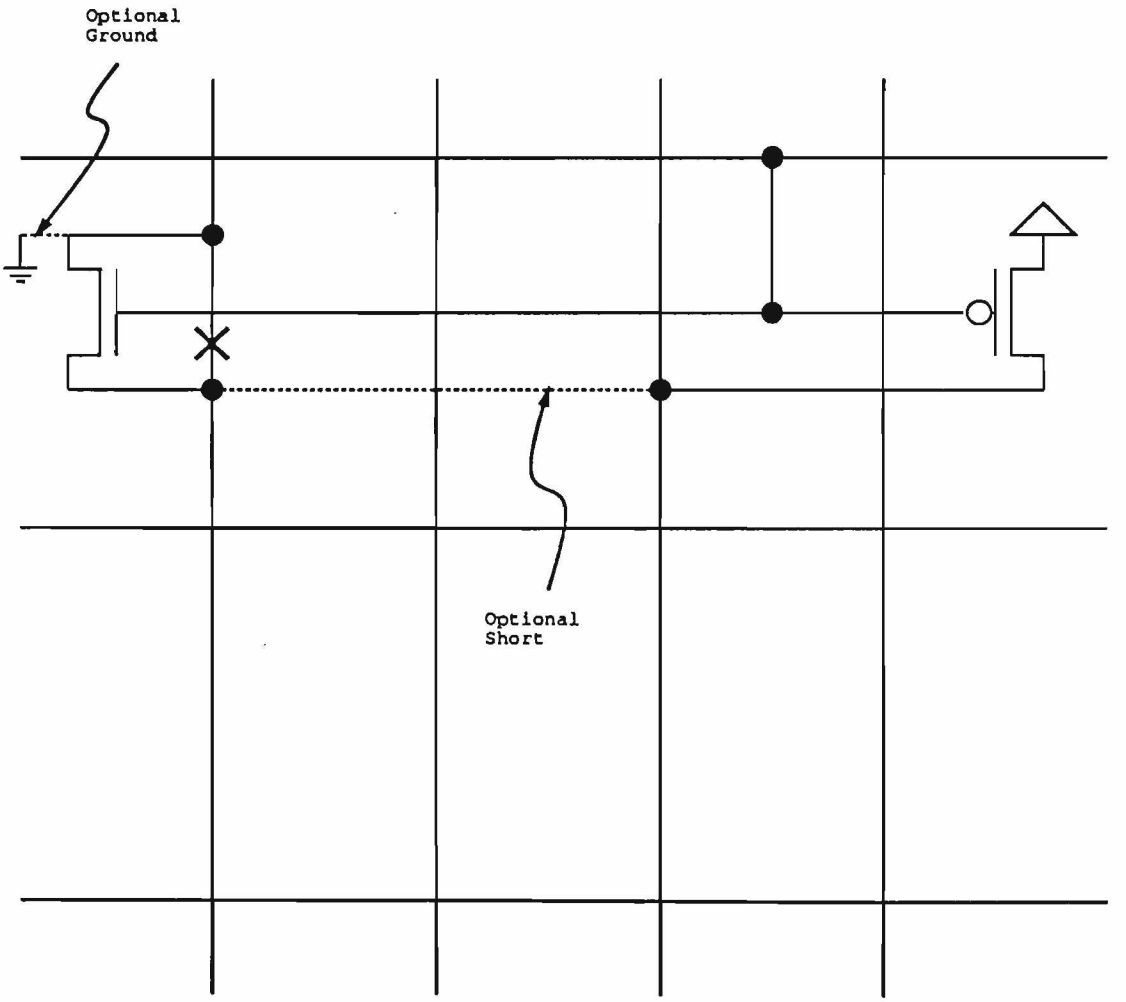
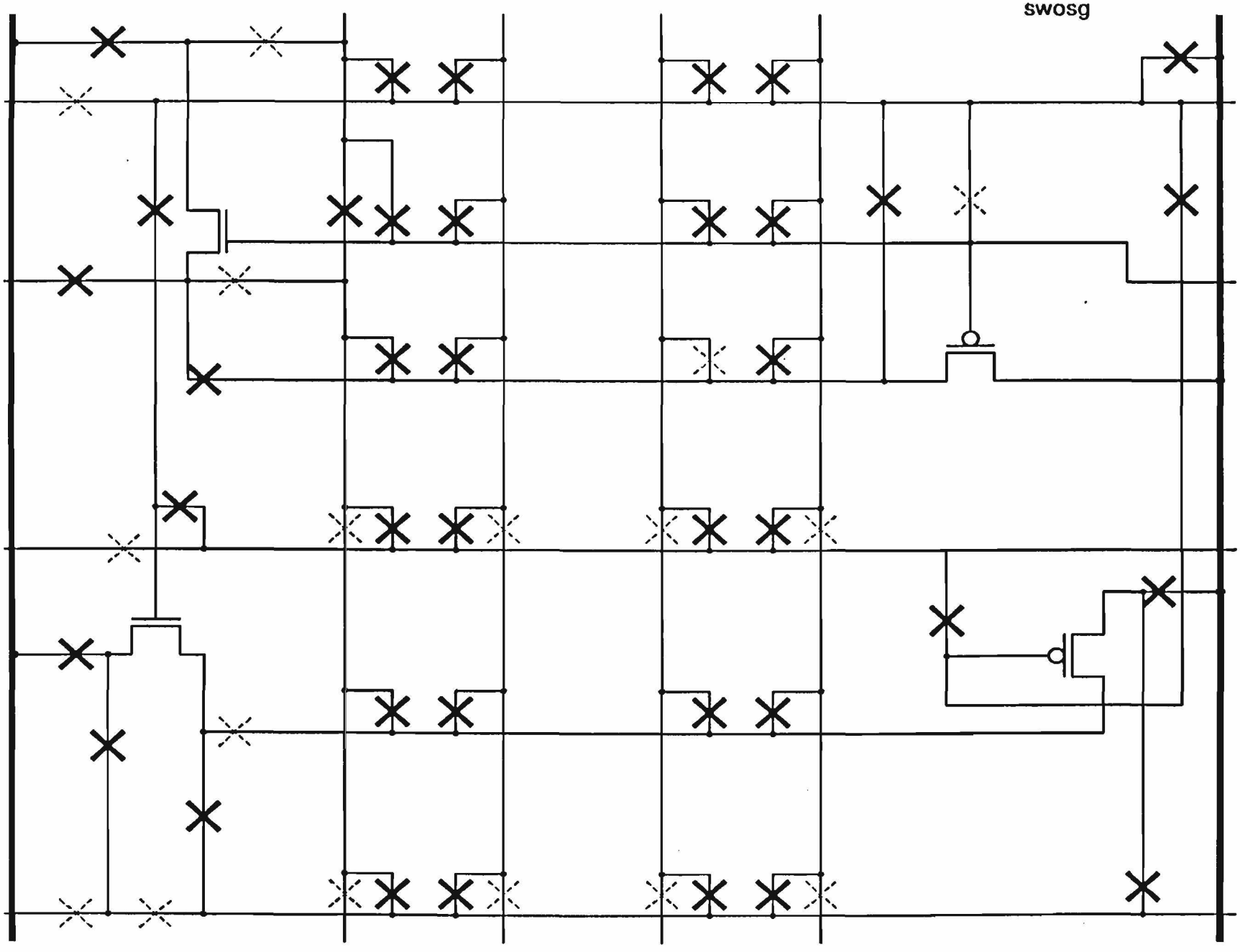
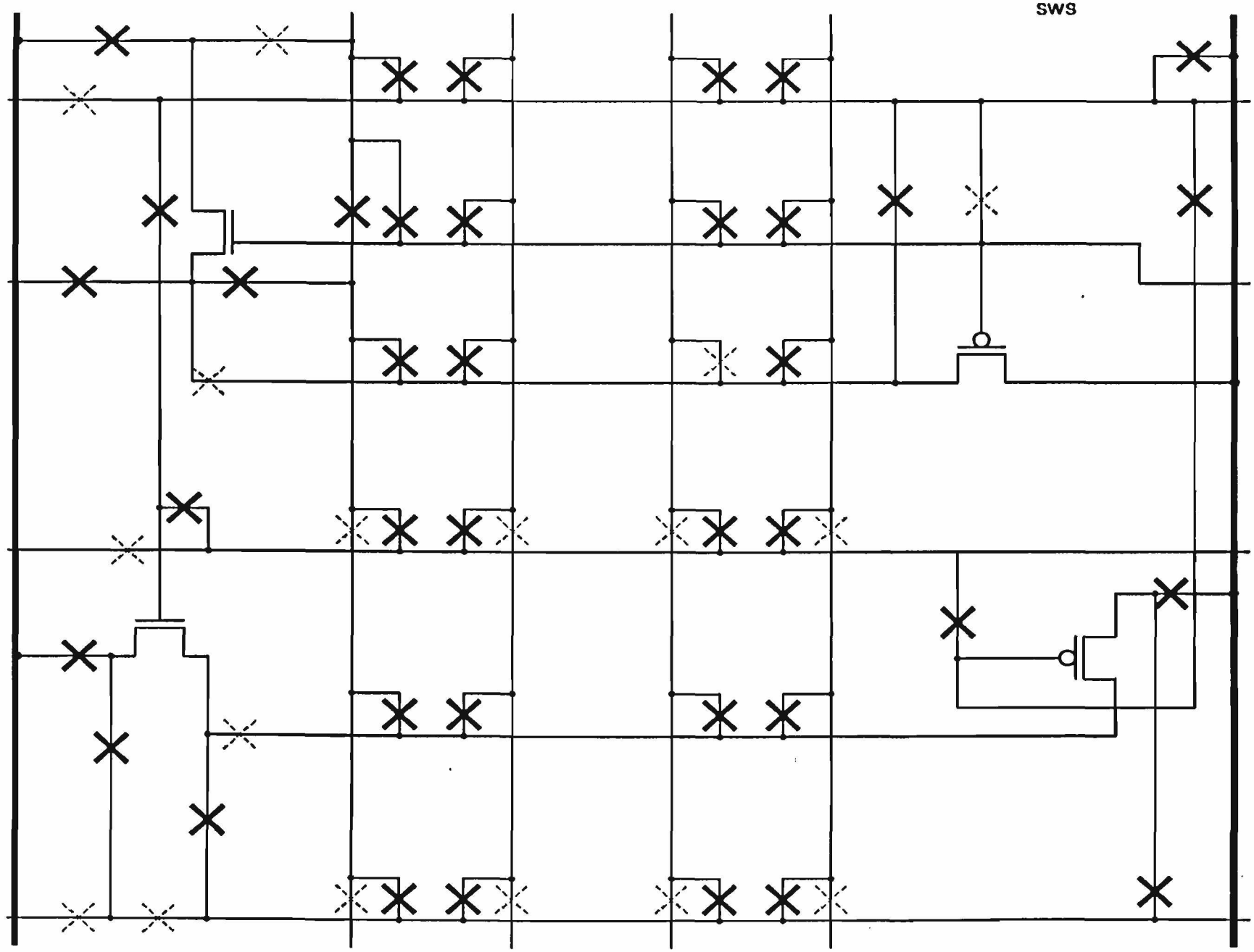
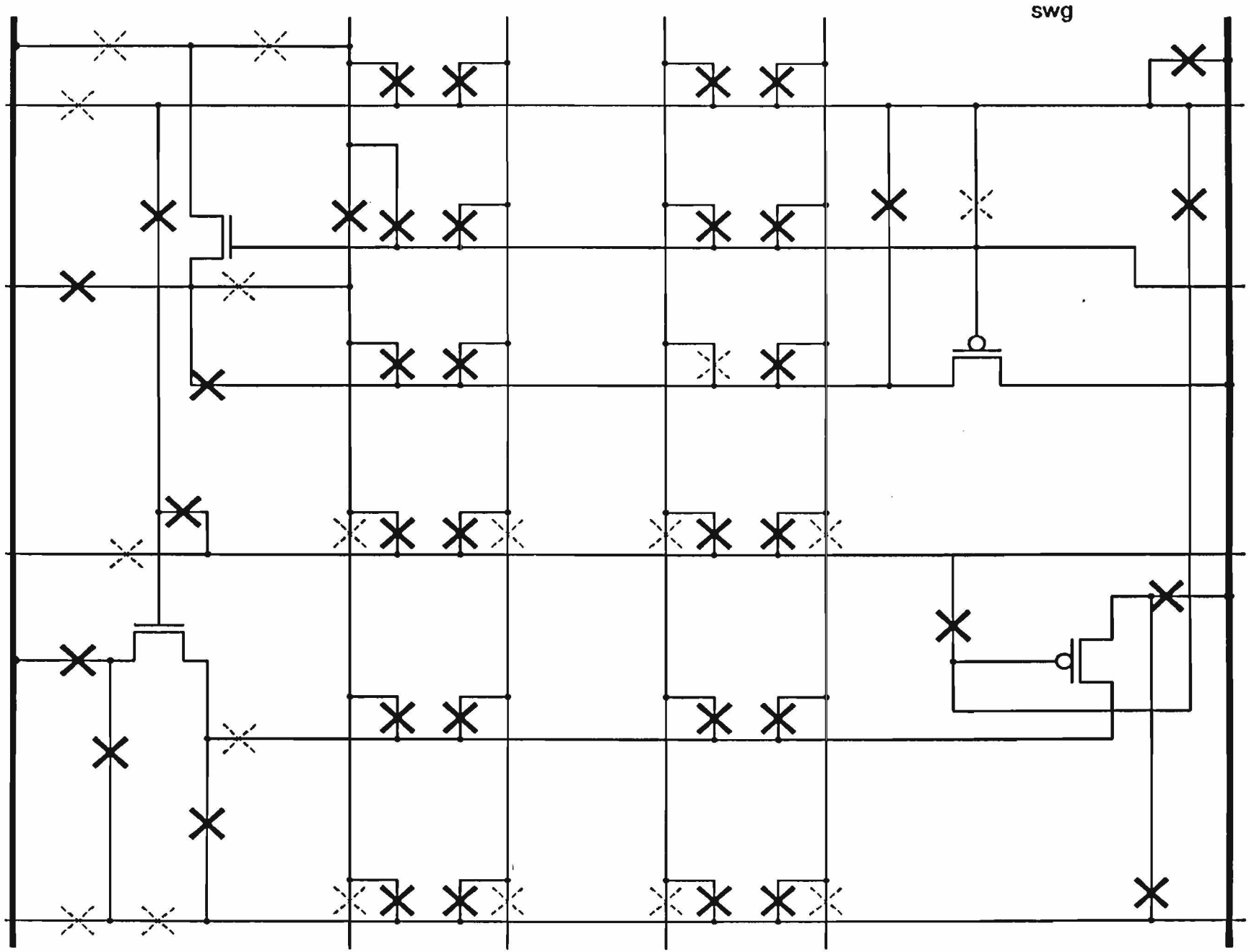
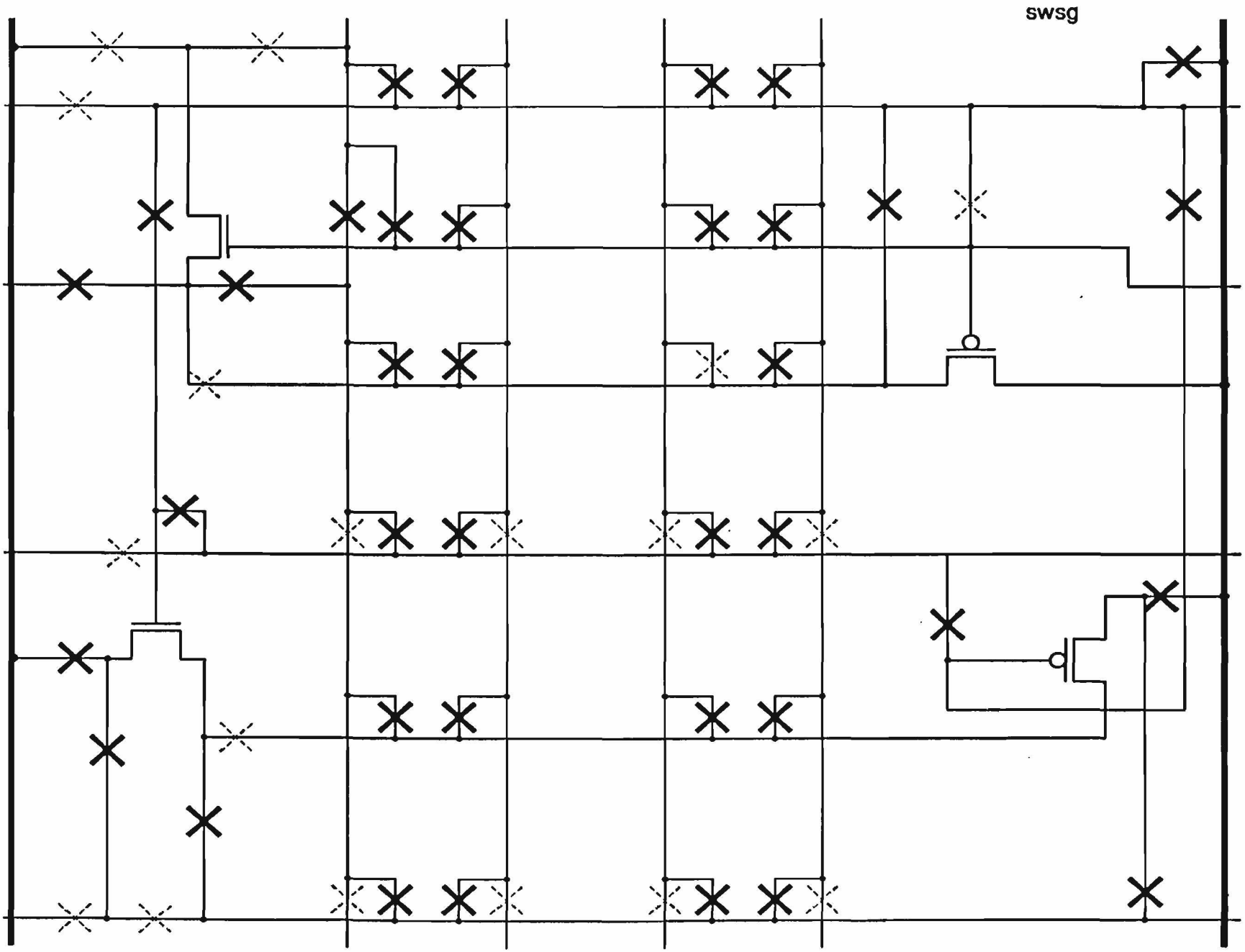


Figure 3.7. The s cell


swosg


SWS


swg


swsg

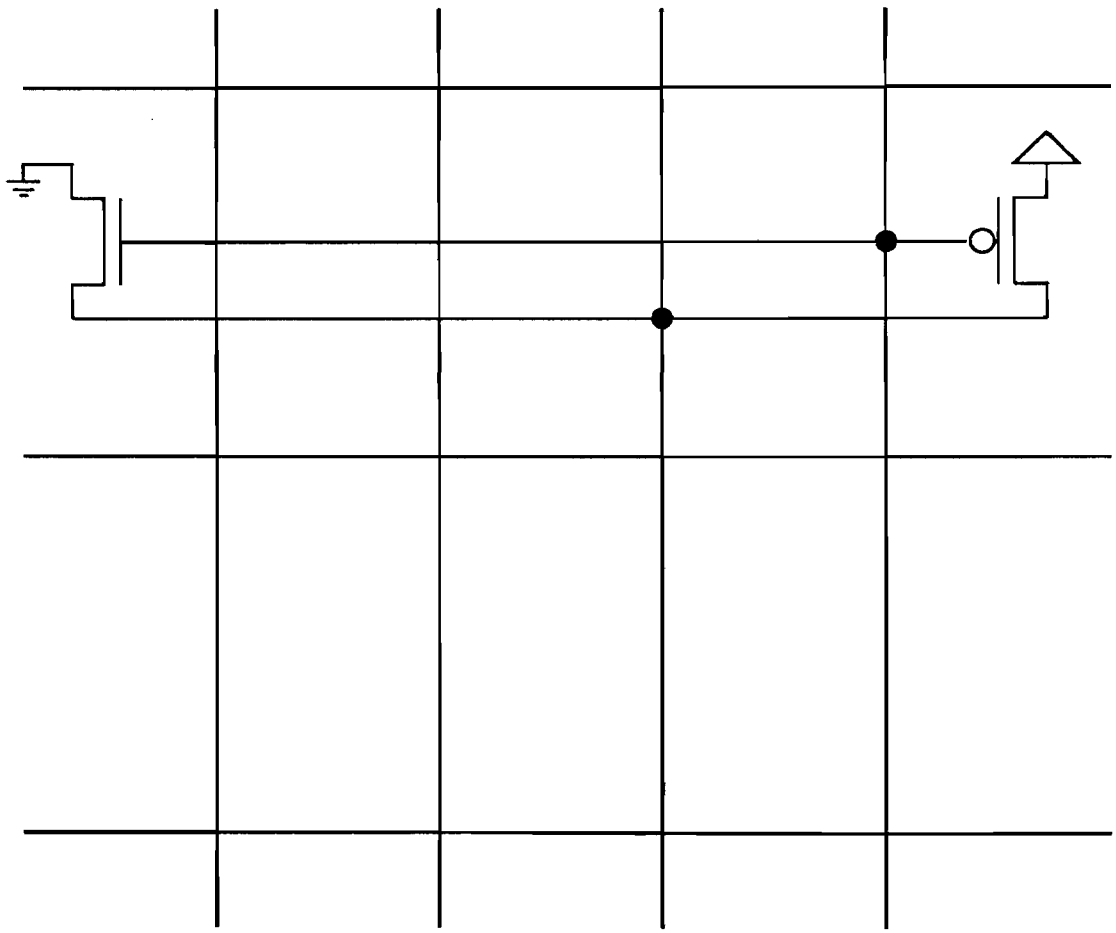
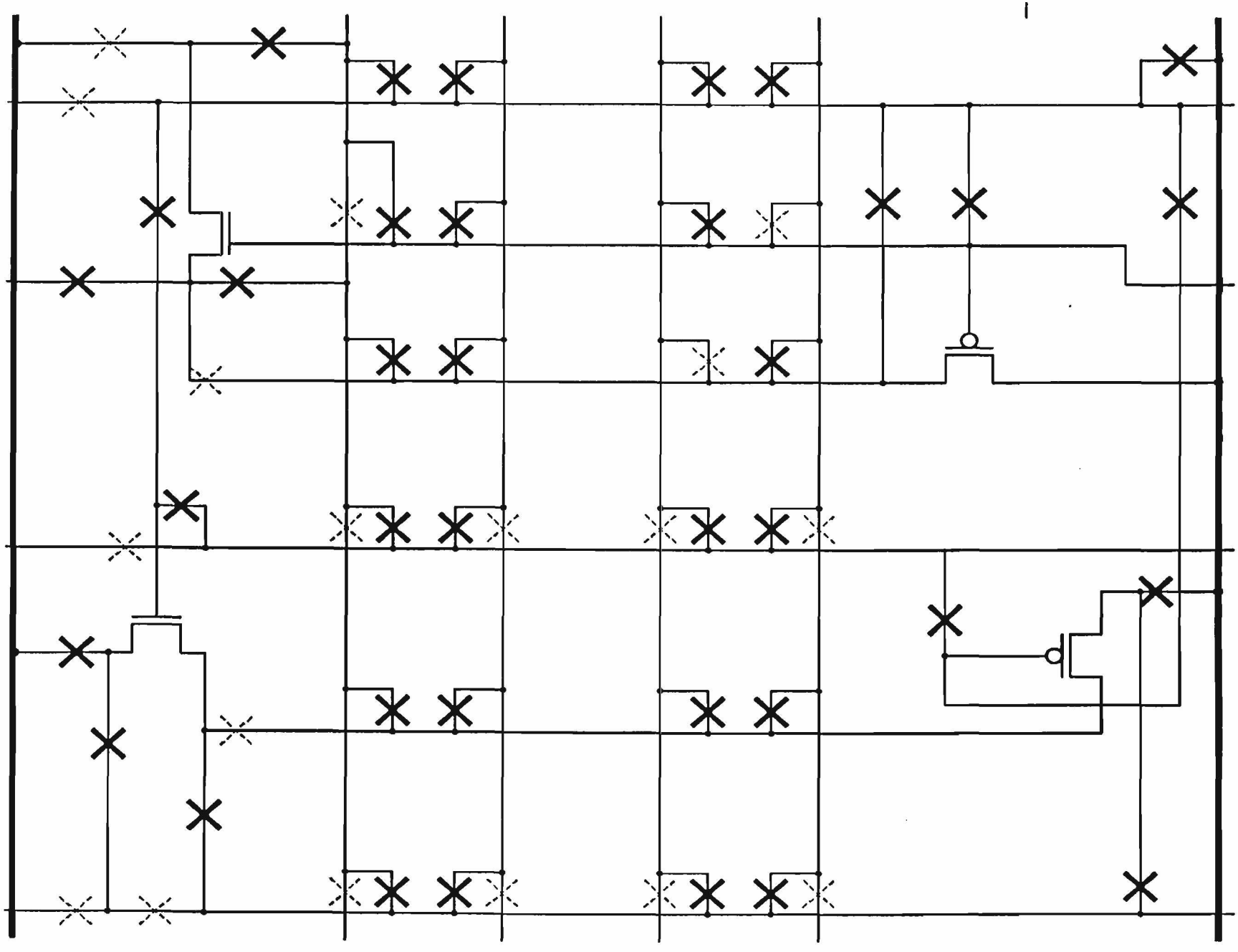


Figure 3.8. The inverter cell (i)



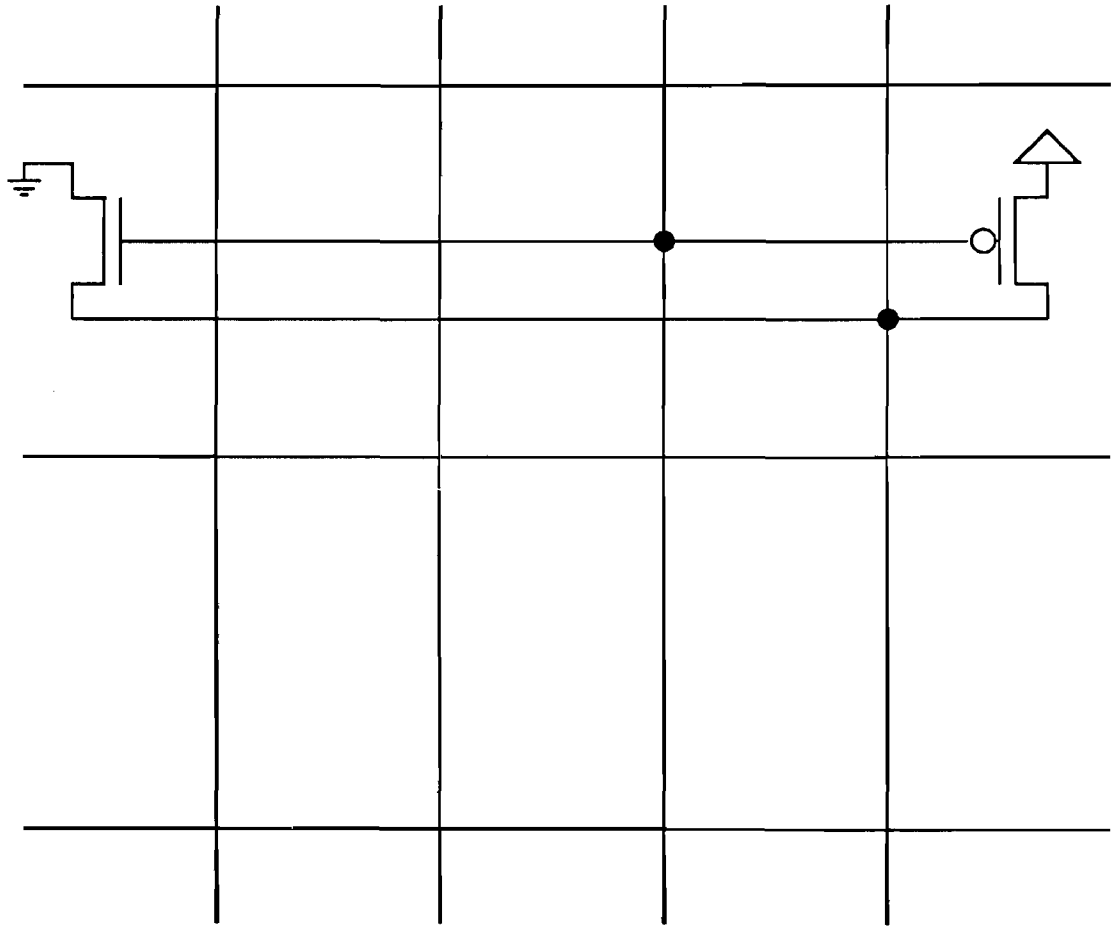
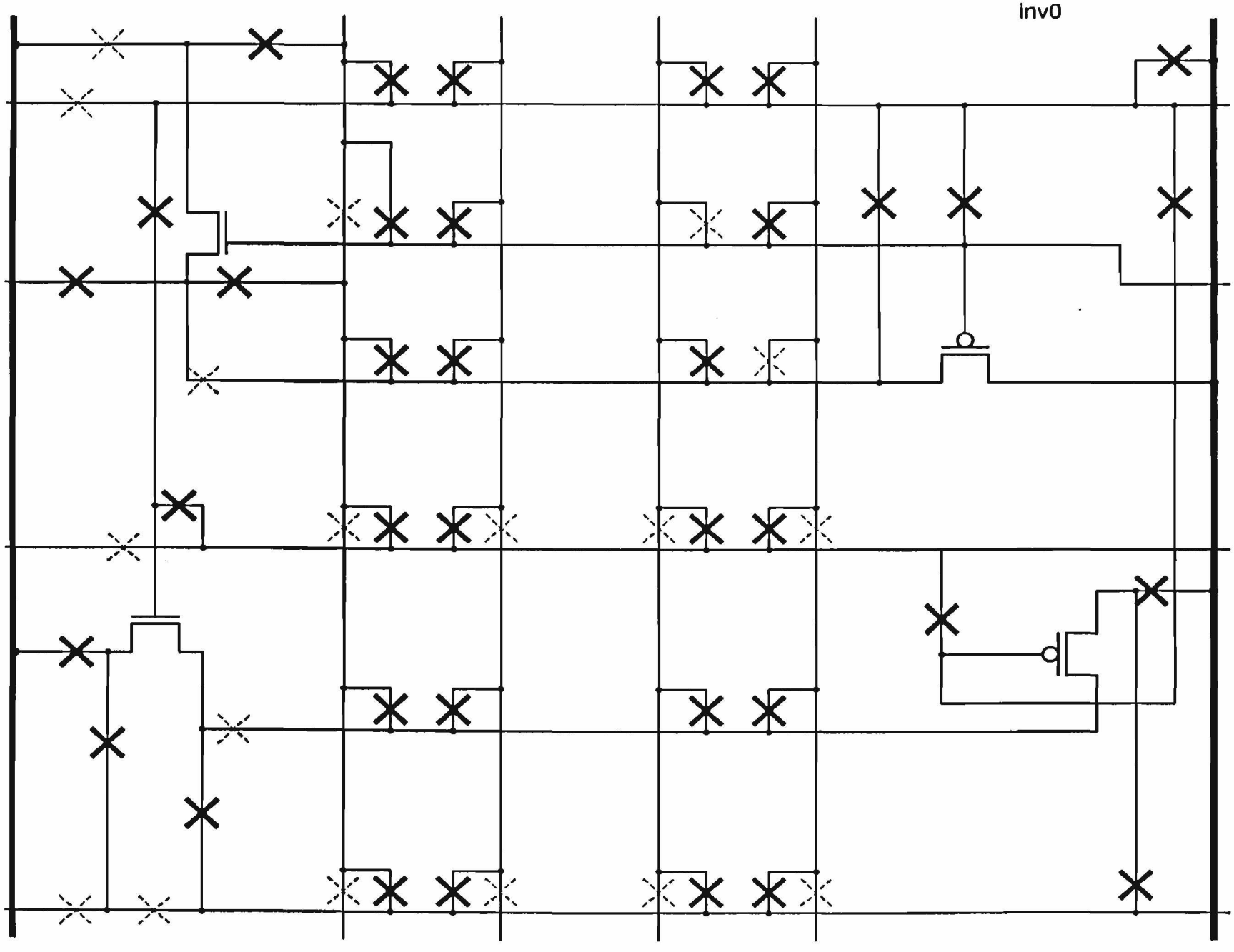


Figure 3.9. The inverter cell (v)


inv0

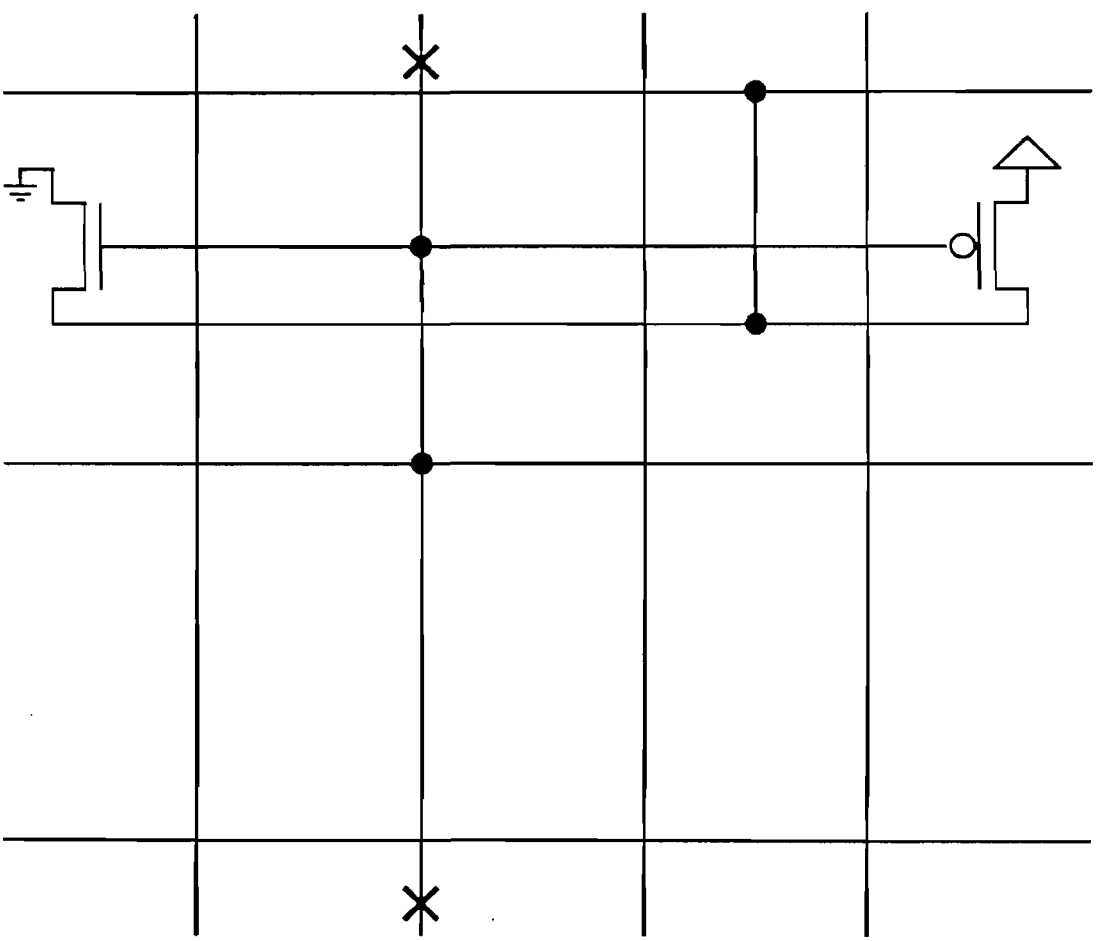
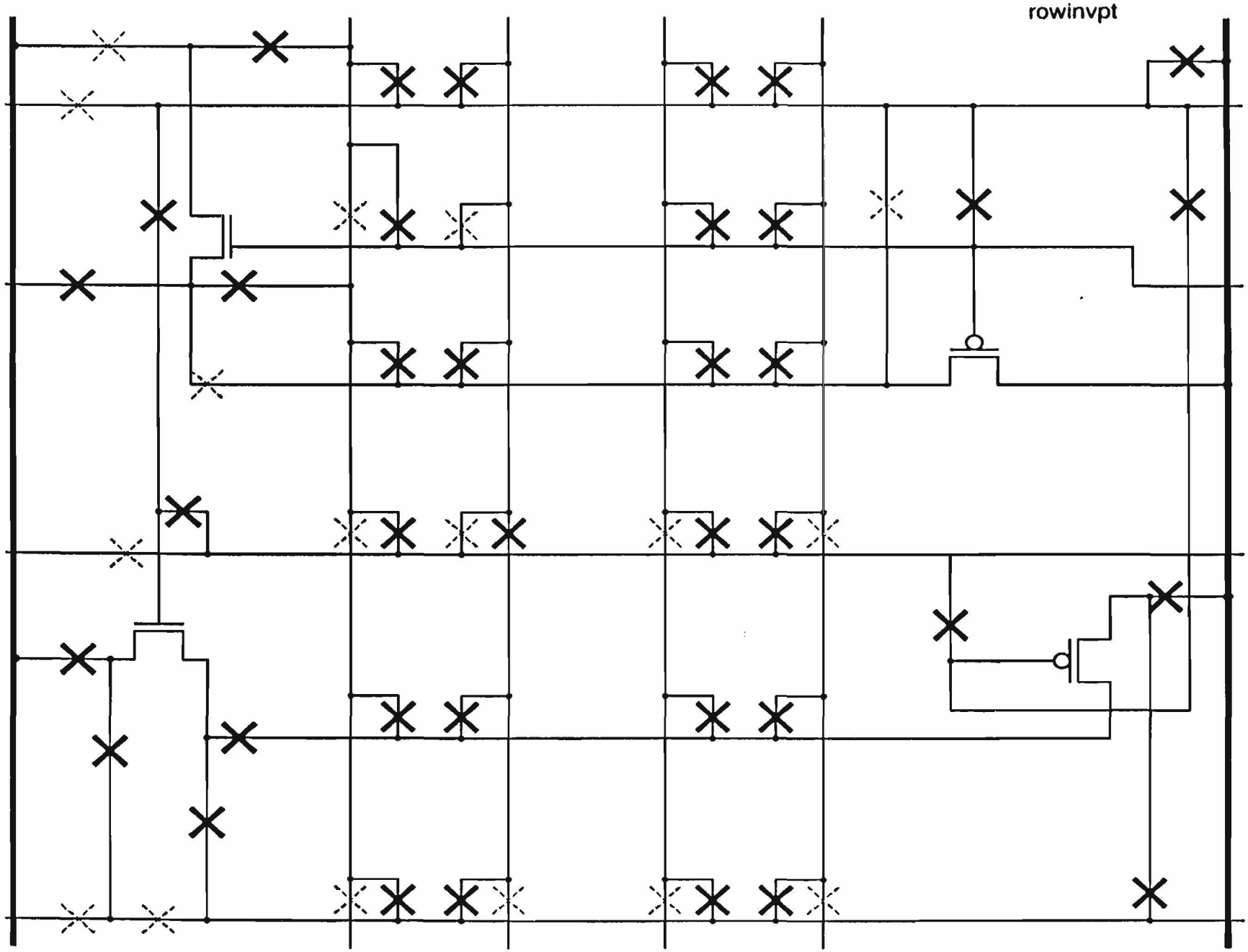


Figure 3.10. The PR - TR inverter cell


rowinvpt

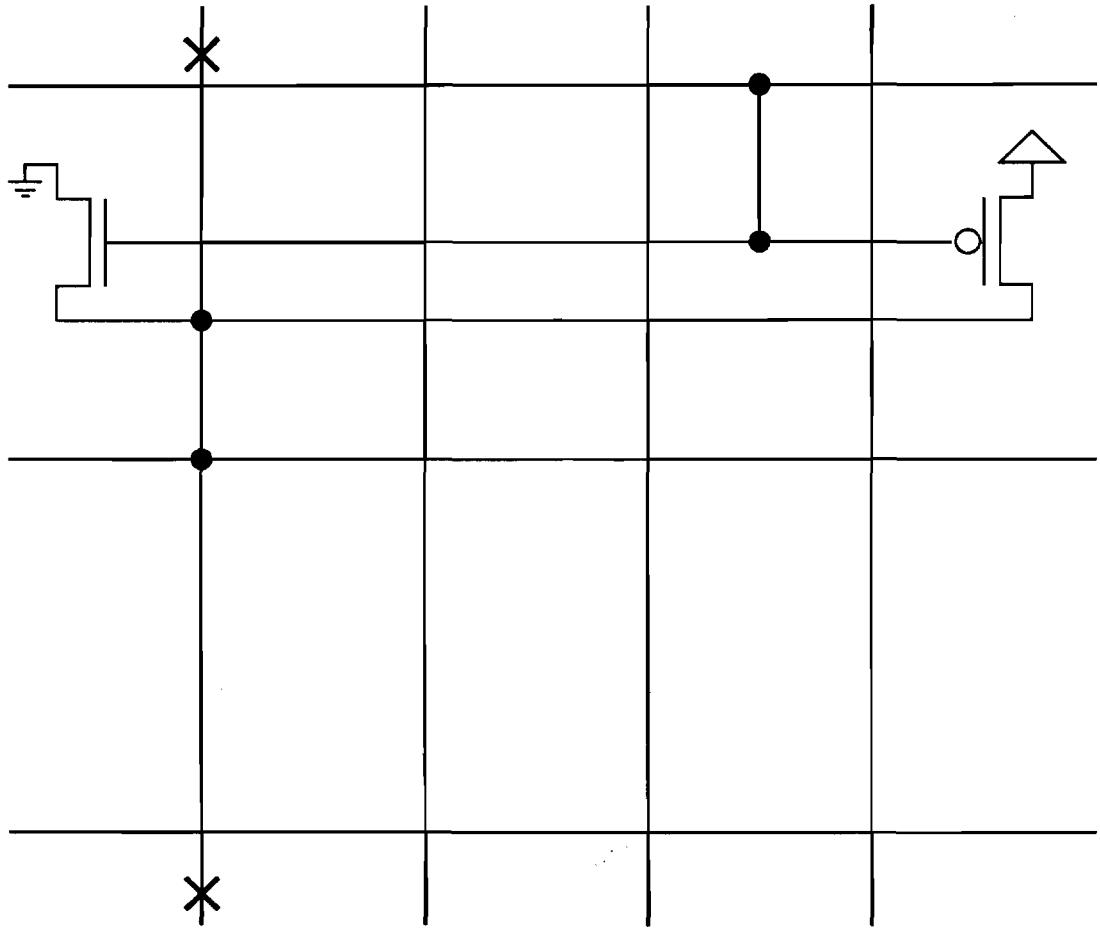
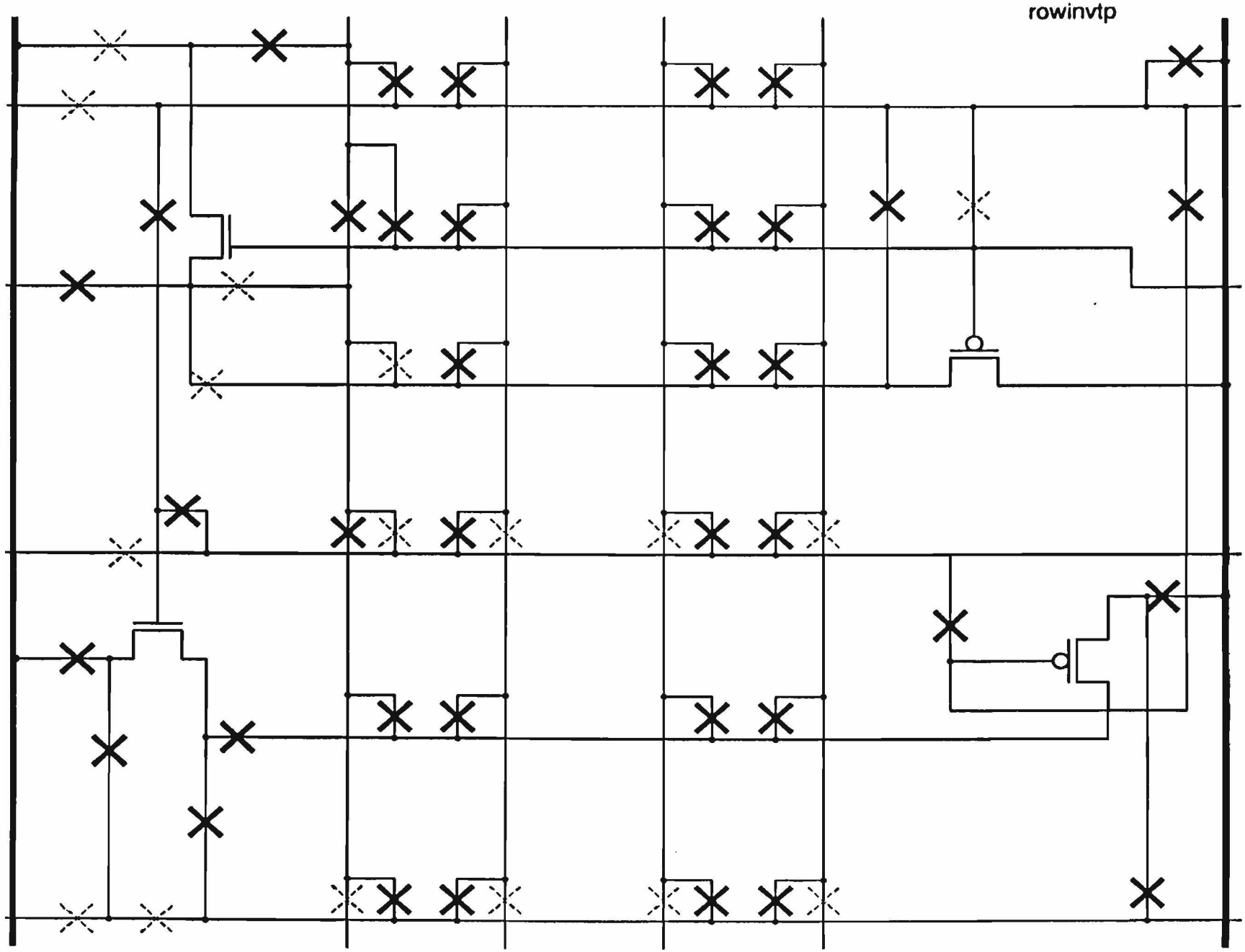


Figure 3.11. The TR - PR inverter cell


rowinvtp

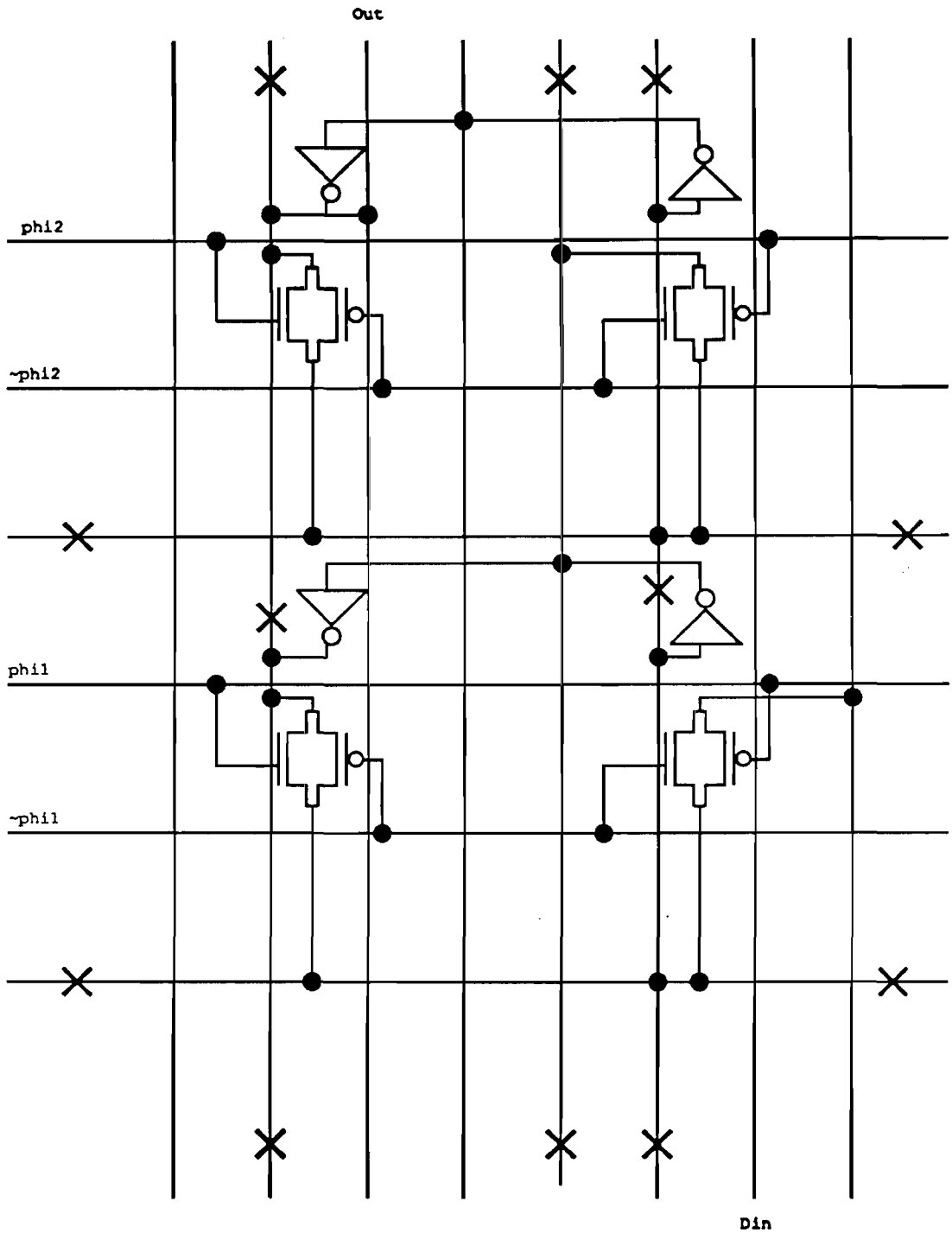
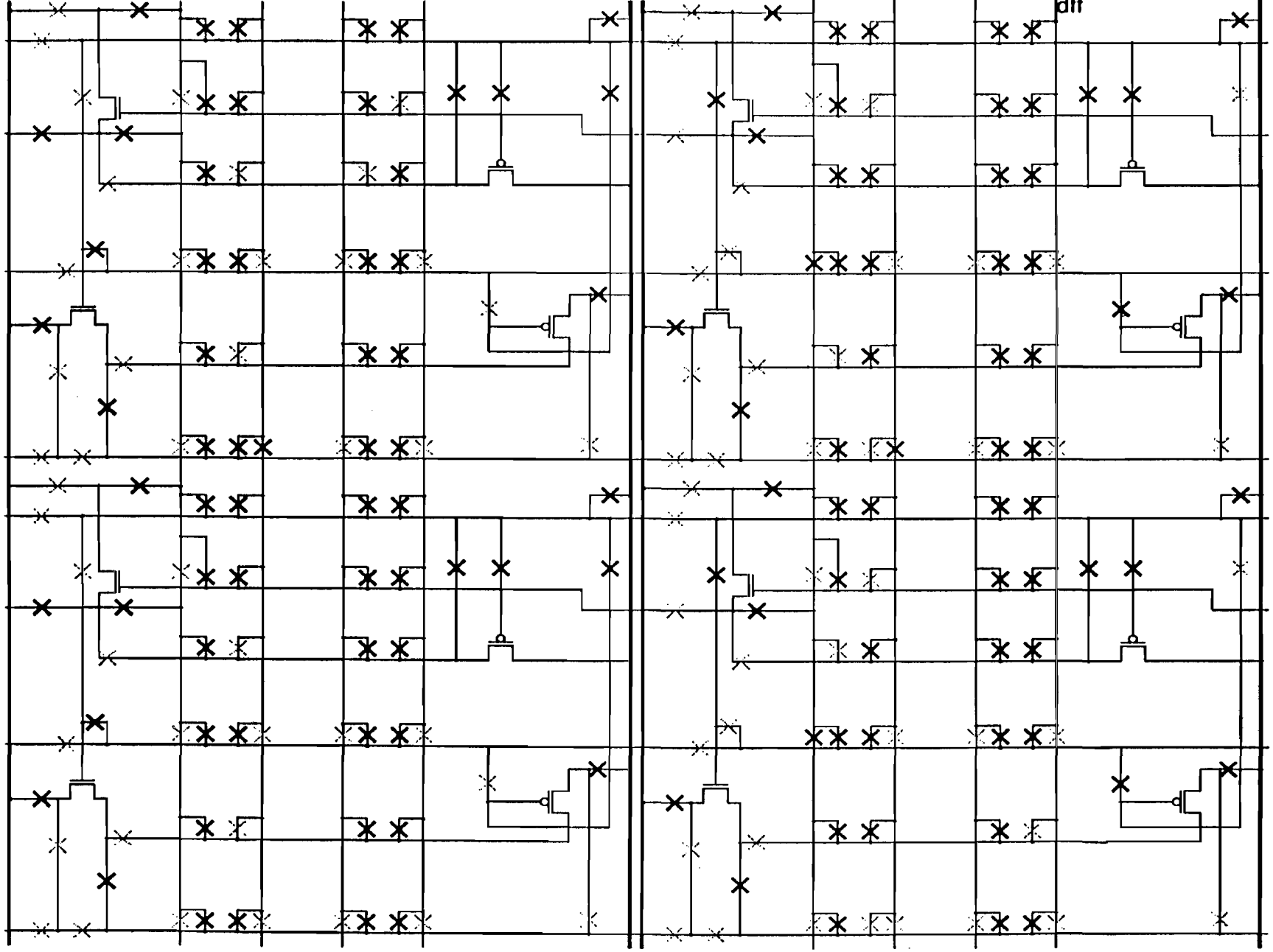


Figure 3.12. The D flip-flop cell


diff

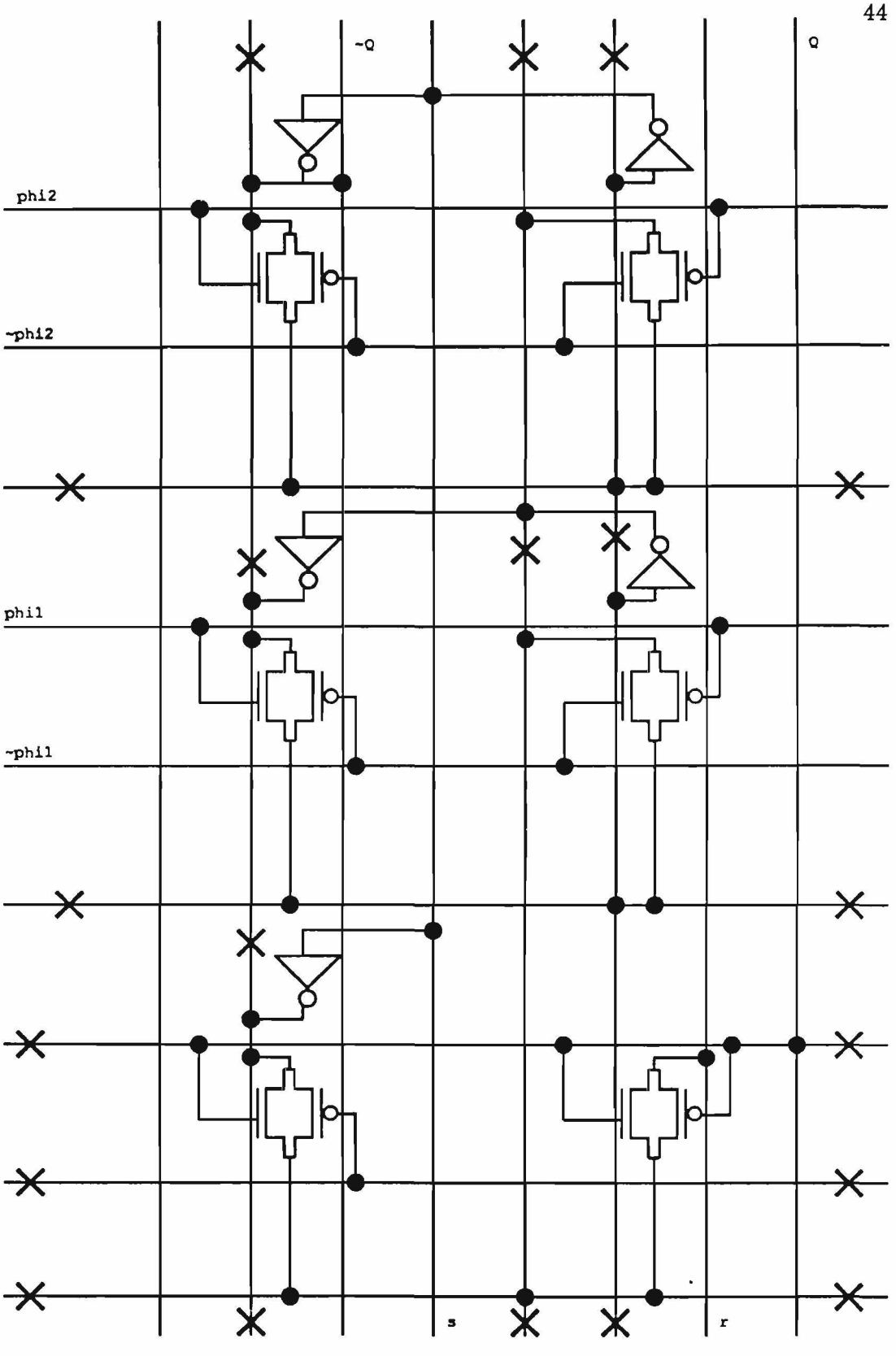
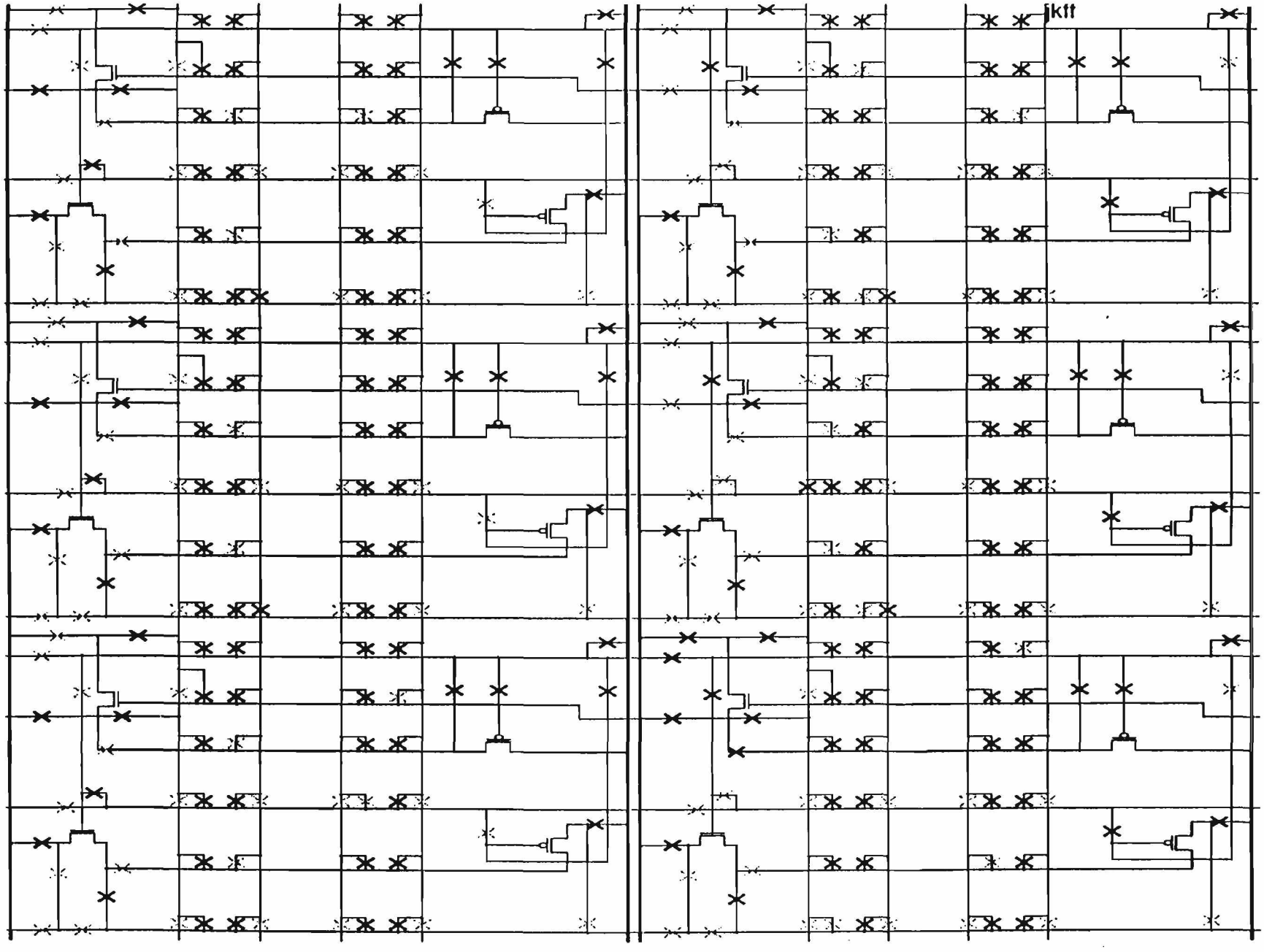


Figure 3.13. The JK flip flop cell


jkff

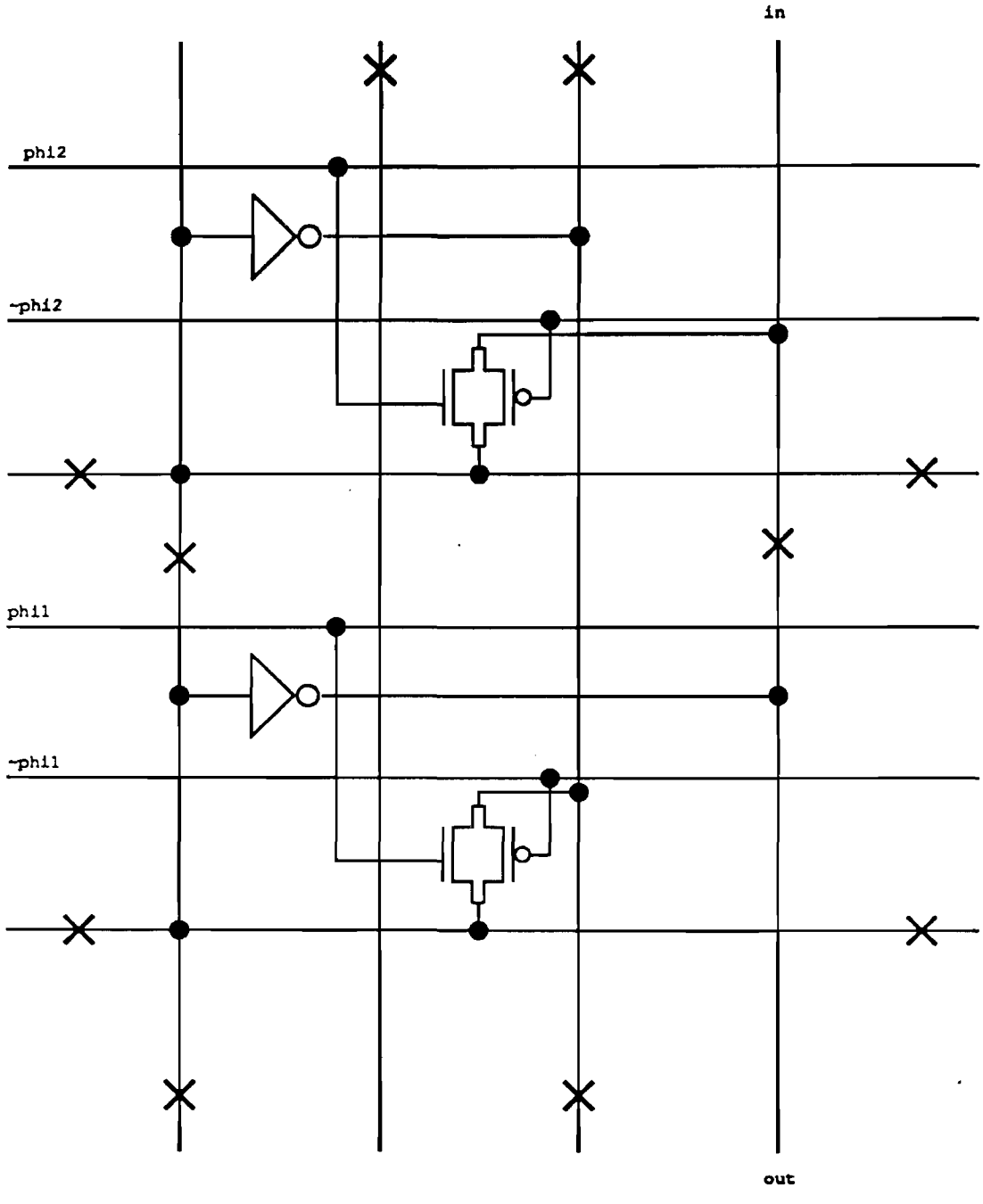
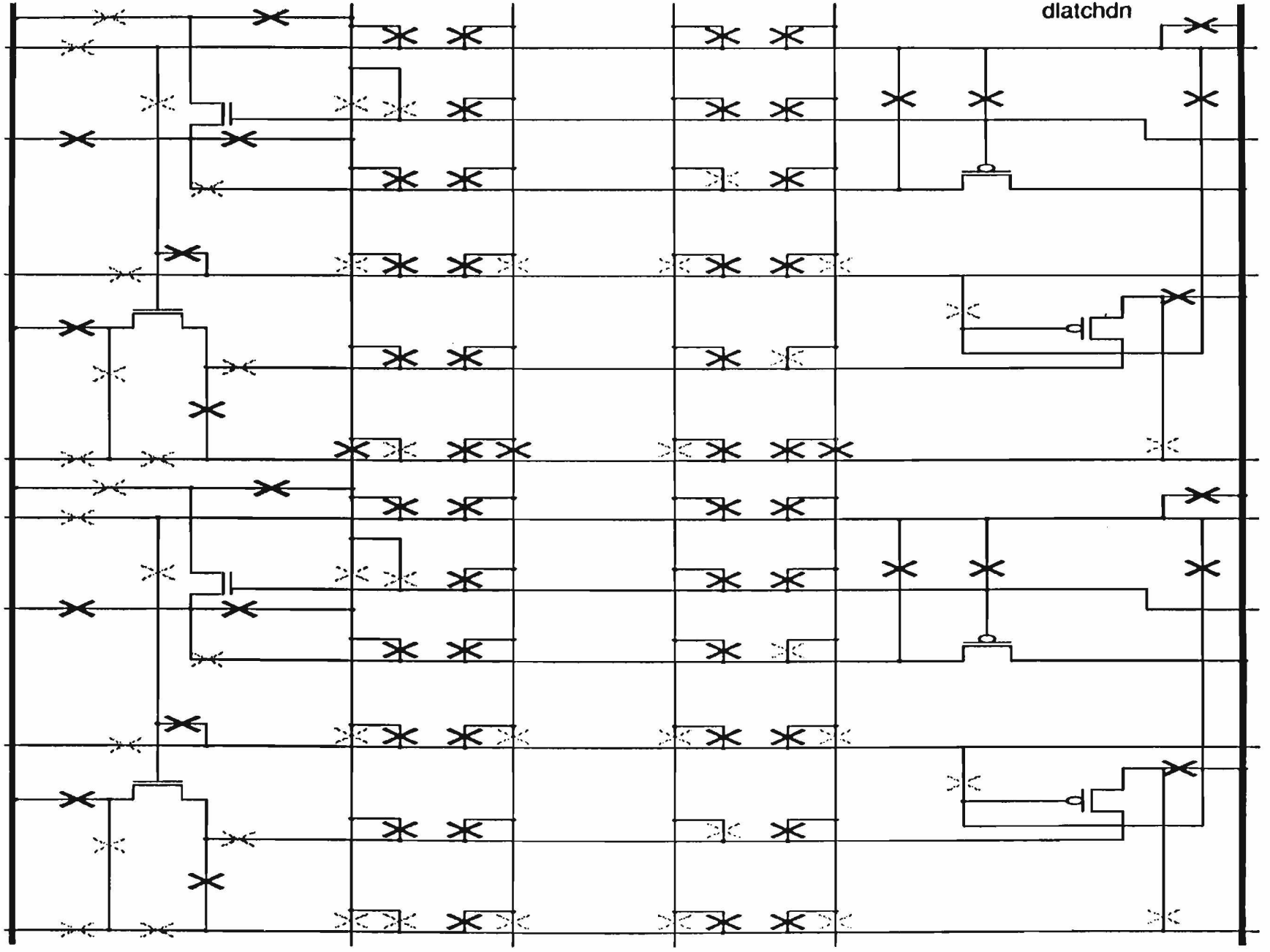


Figure 3.14. The dynamic latch cell (down)


dlatchdn

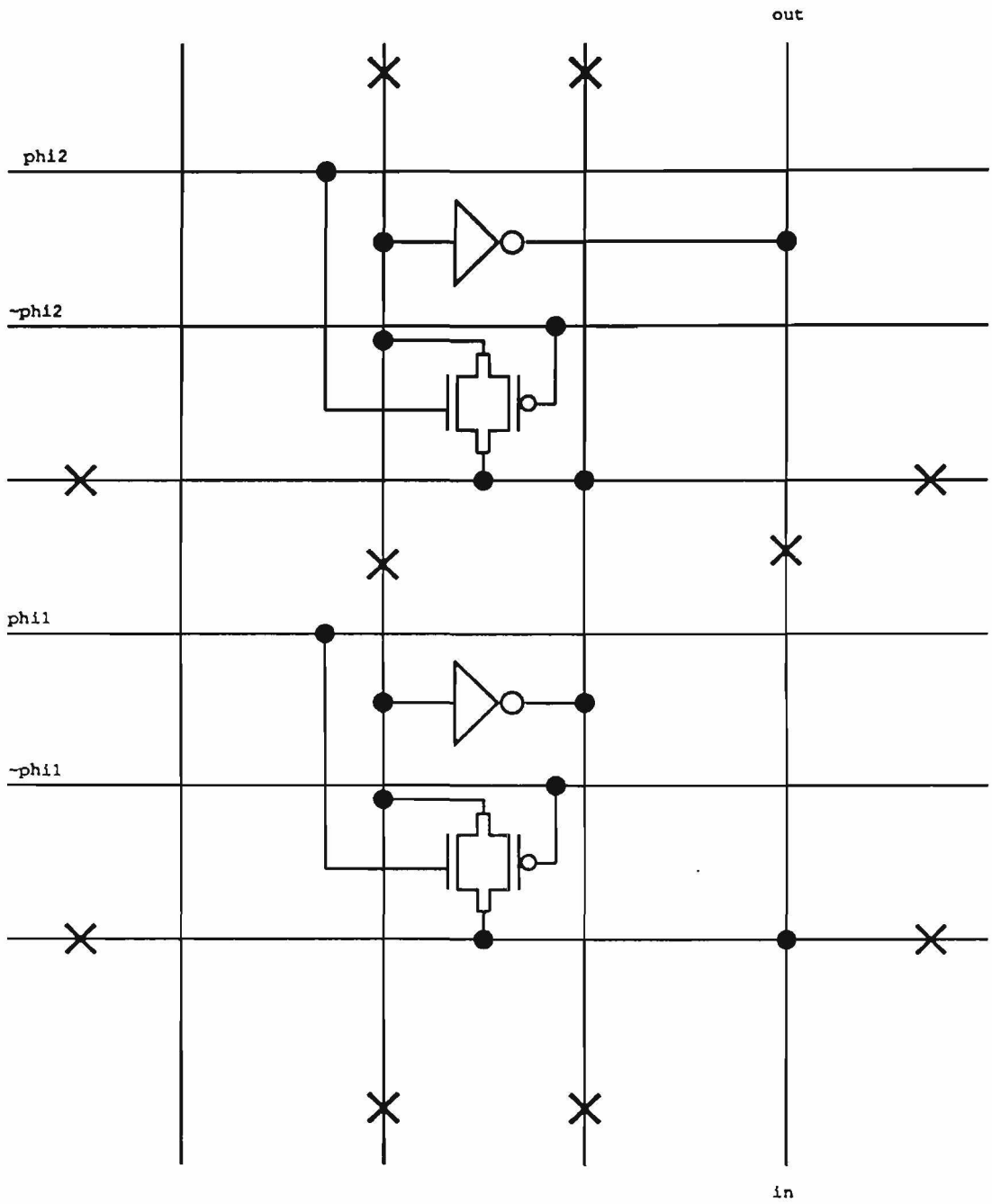
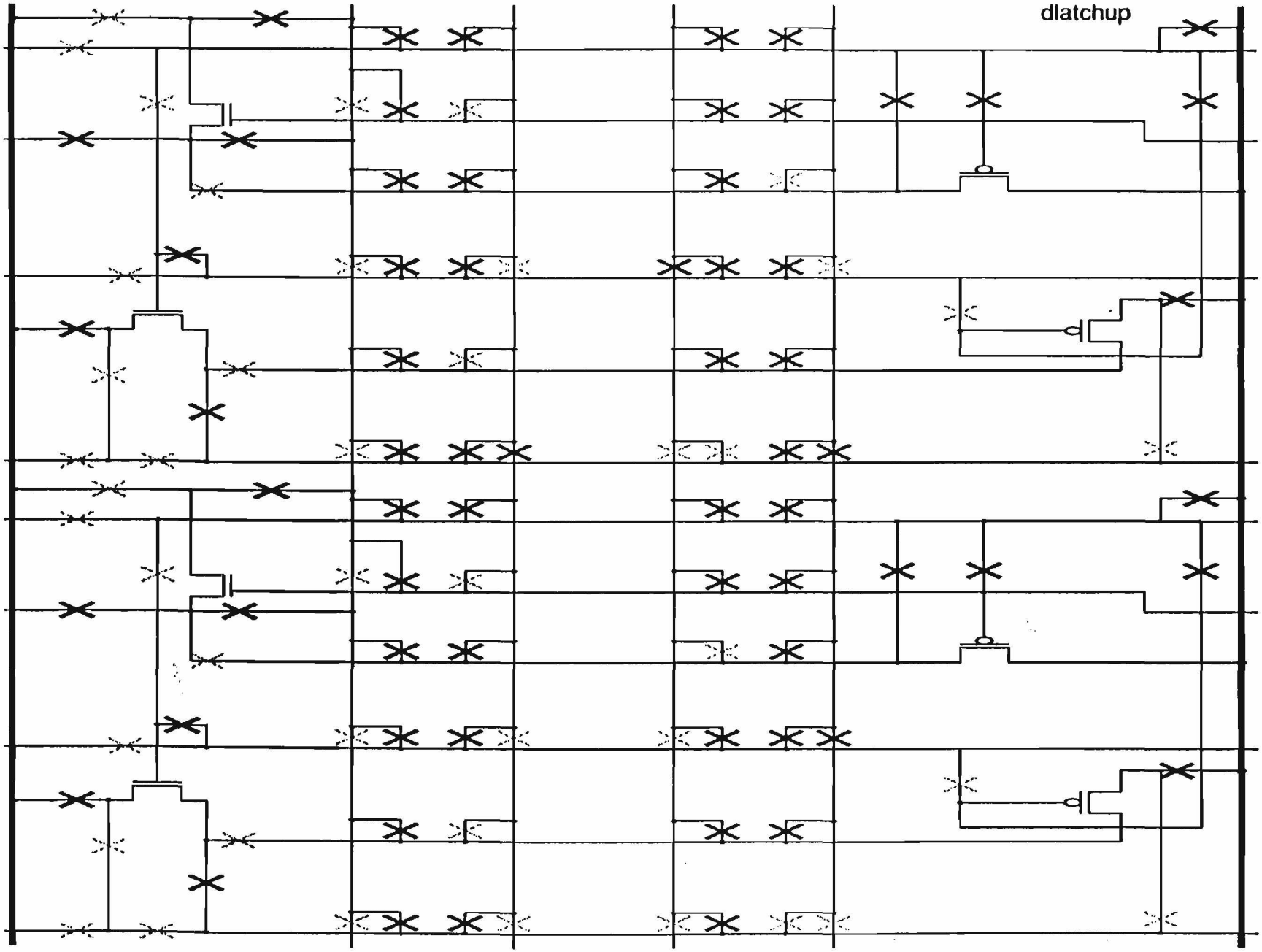


Figure 3.15. The dynamic latch cell (up)


dlatchup

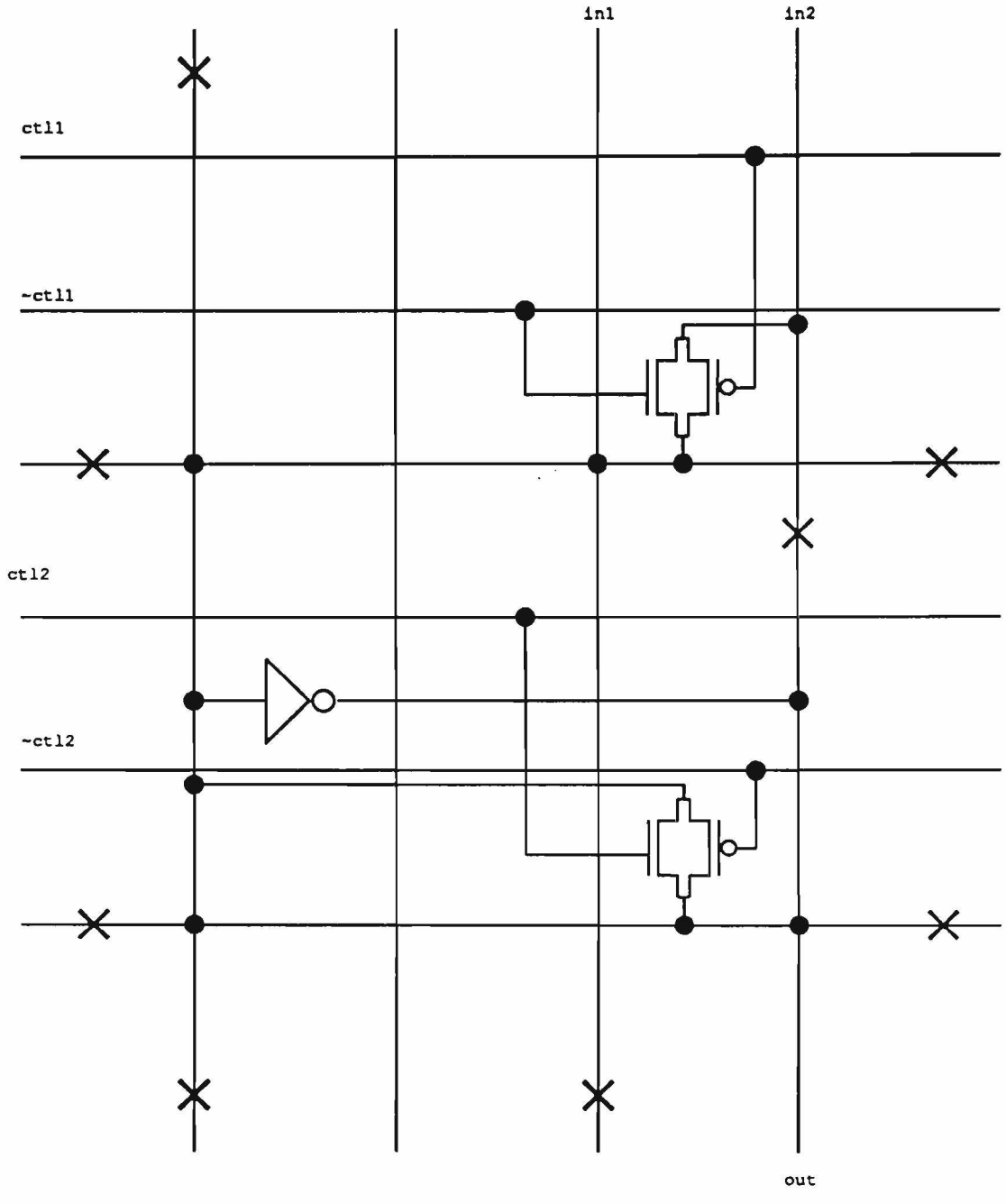
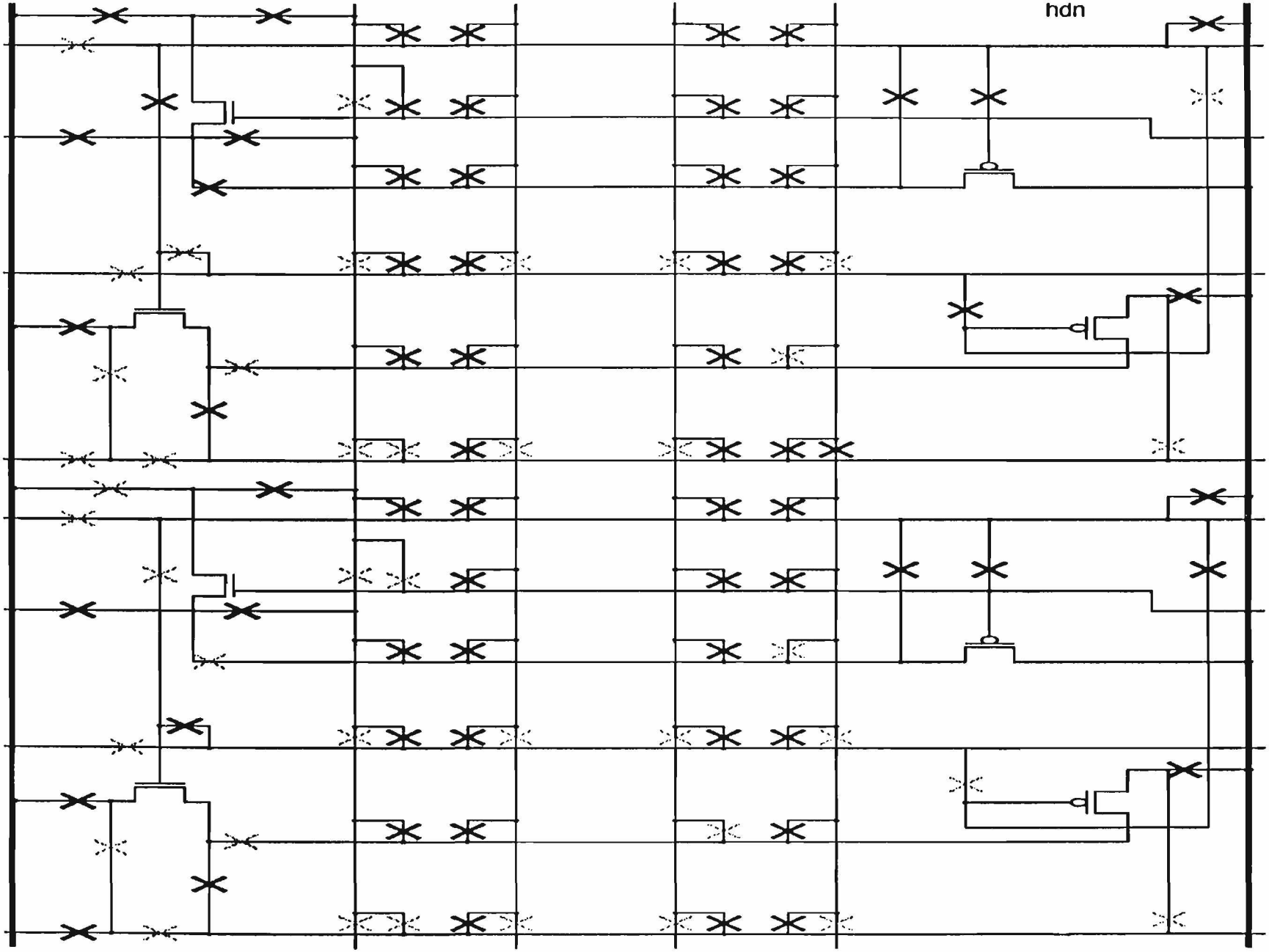


Figure 3.16. The multiplexor cell (down)


hdn

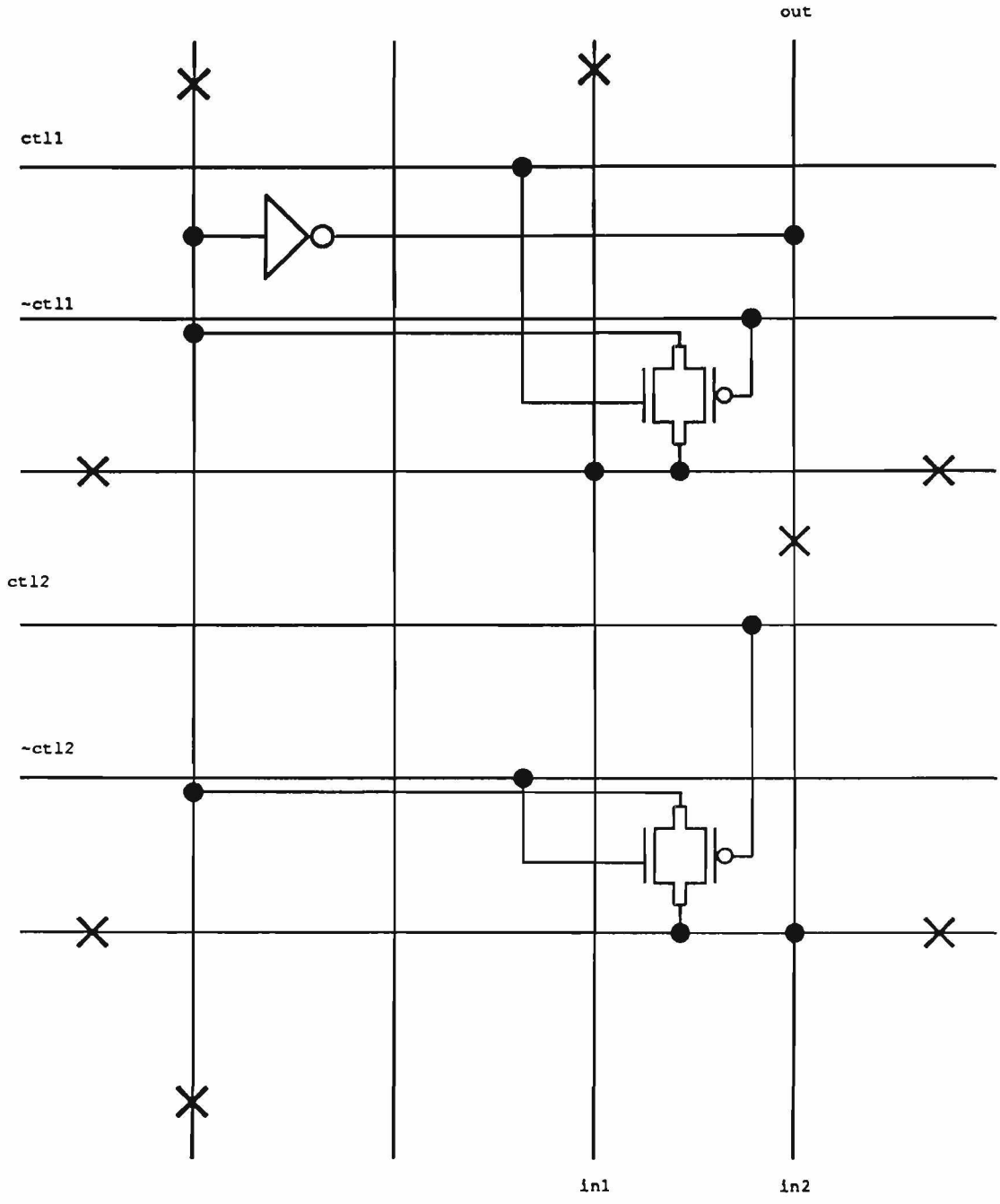
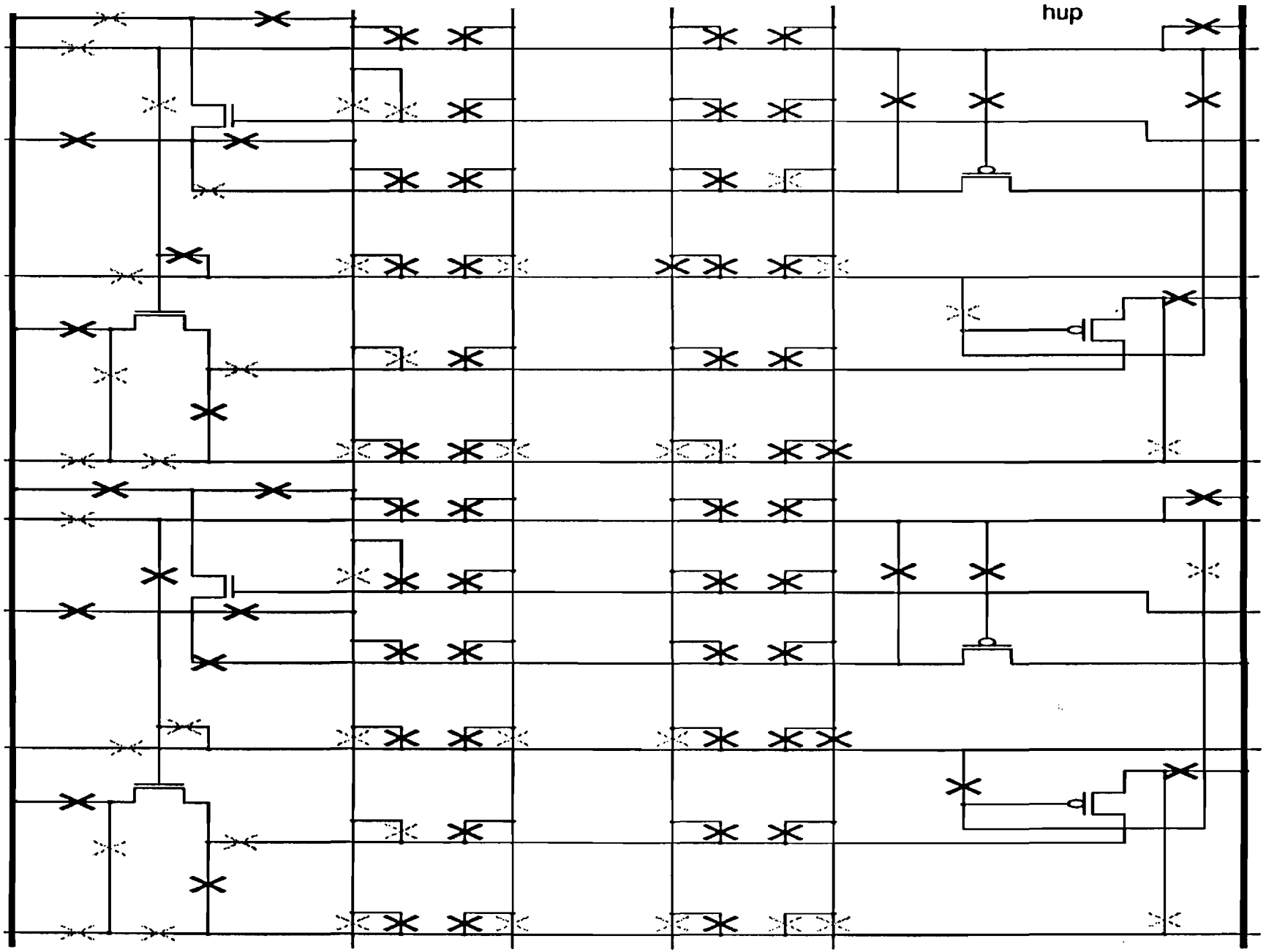


Figure 3.17. The multiplexor cell (up)


hup

3.1.7 Power Cells

Figure 3.18 and Figure 3.19 are the power cells.

3.1.8 Break Cells

Figure 3.20 shows all of the possible breaks.

3.1.9 Connection Cells

Figure 3.21 shows all of the possible connections.

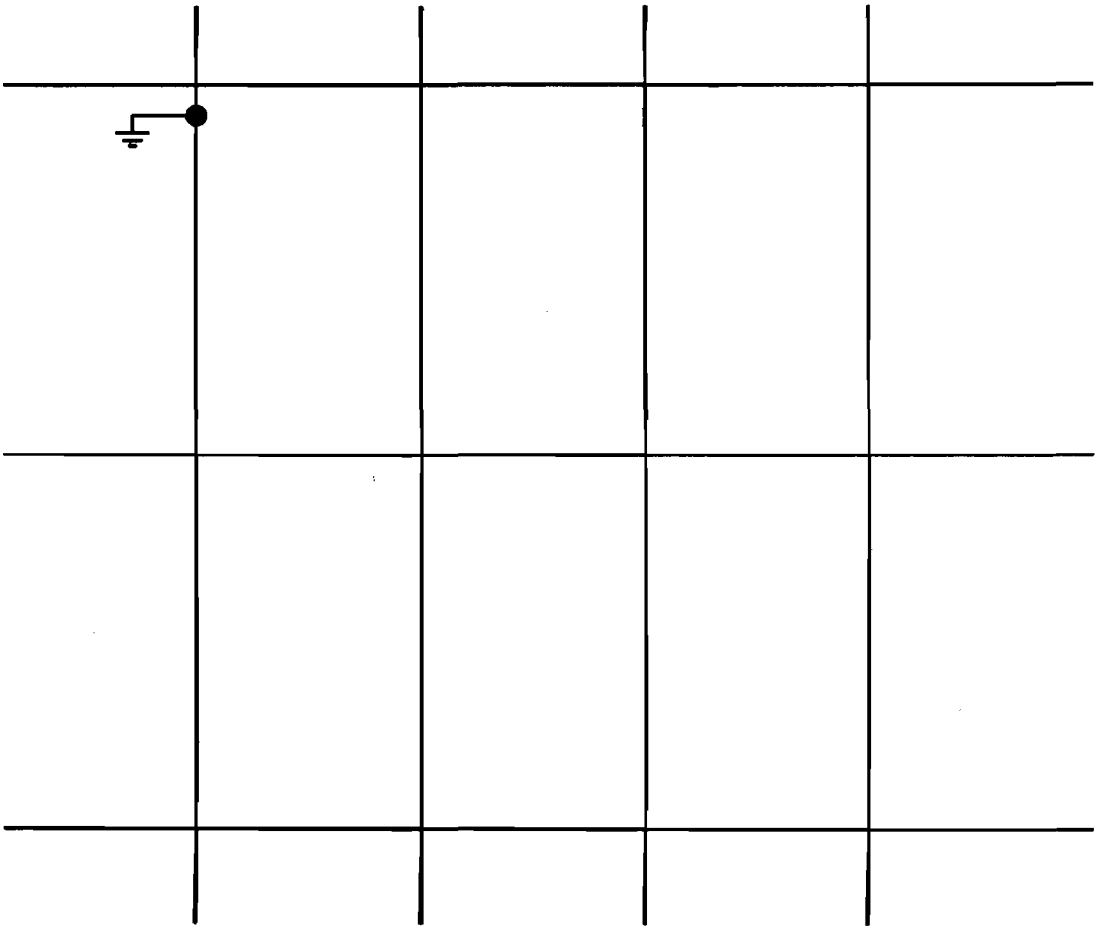
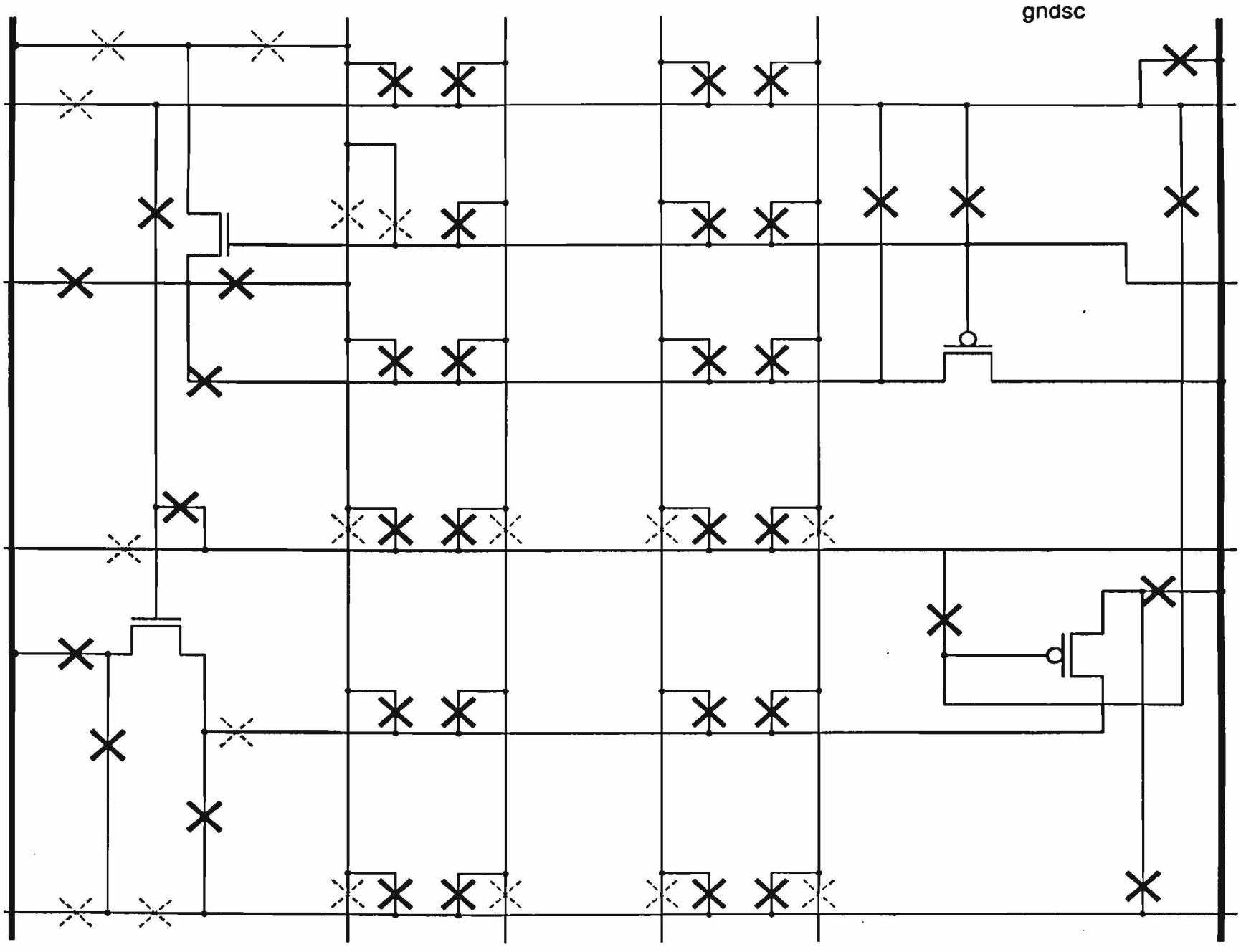


Figure 3.18. The ground cell)


gndsc

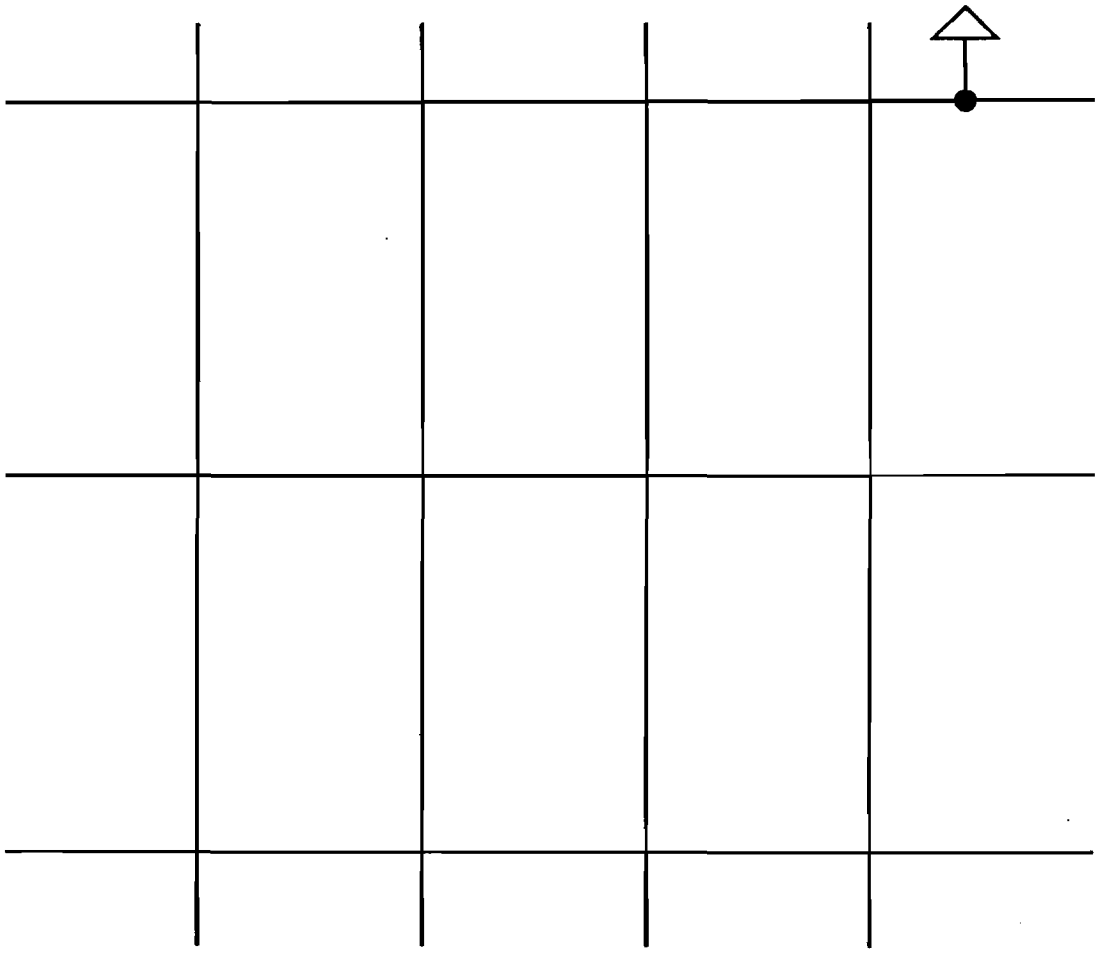
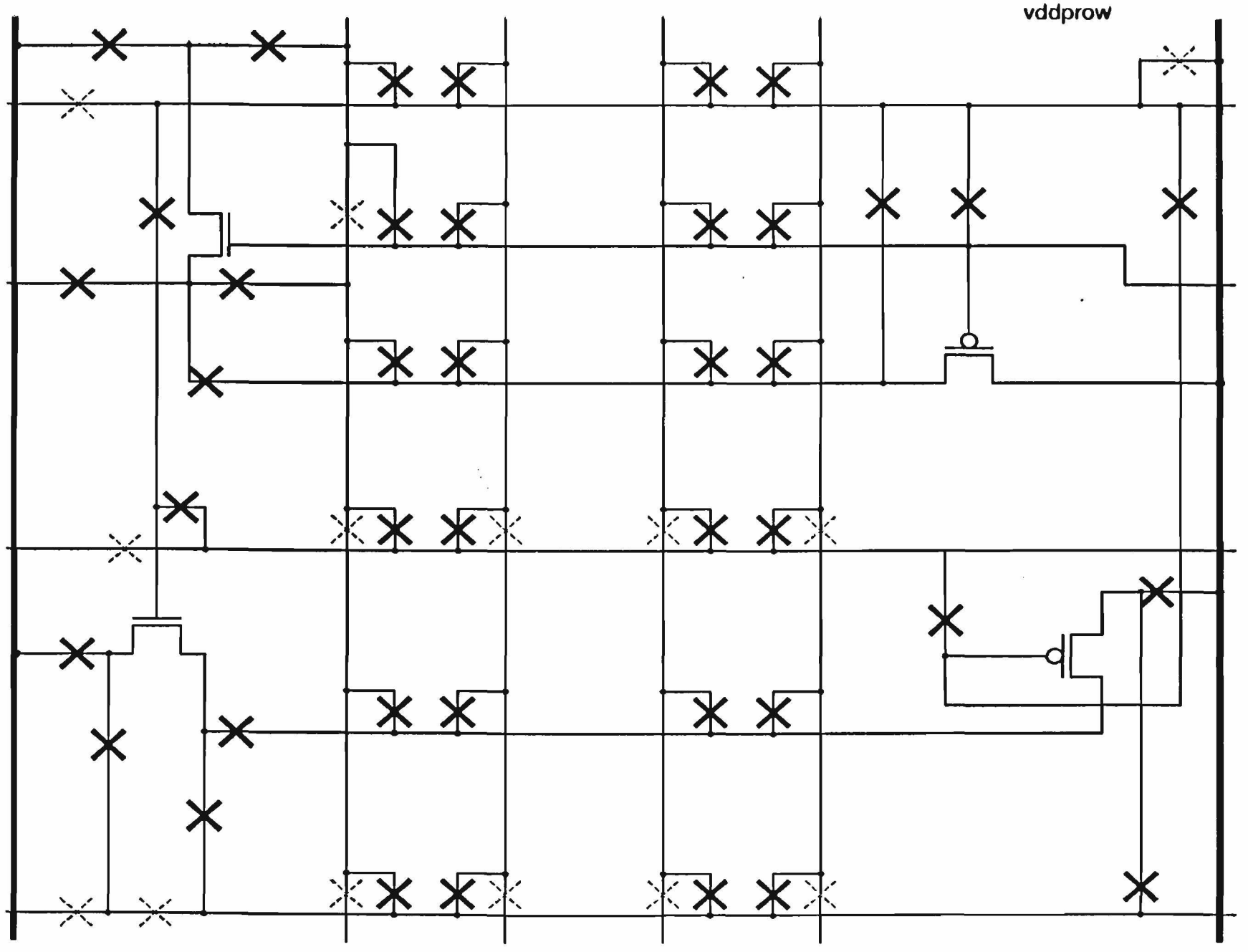


Figure 3.19. The VDD cell


vddprow

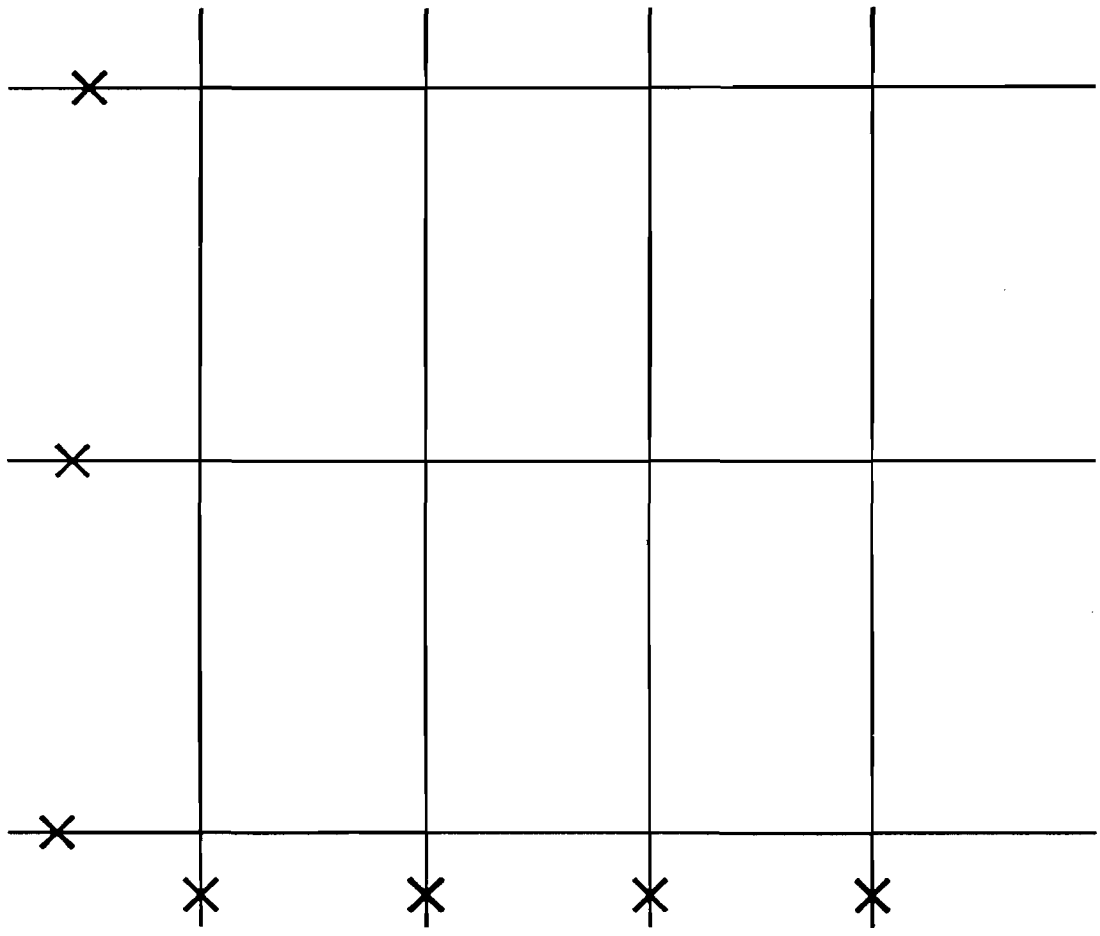
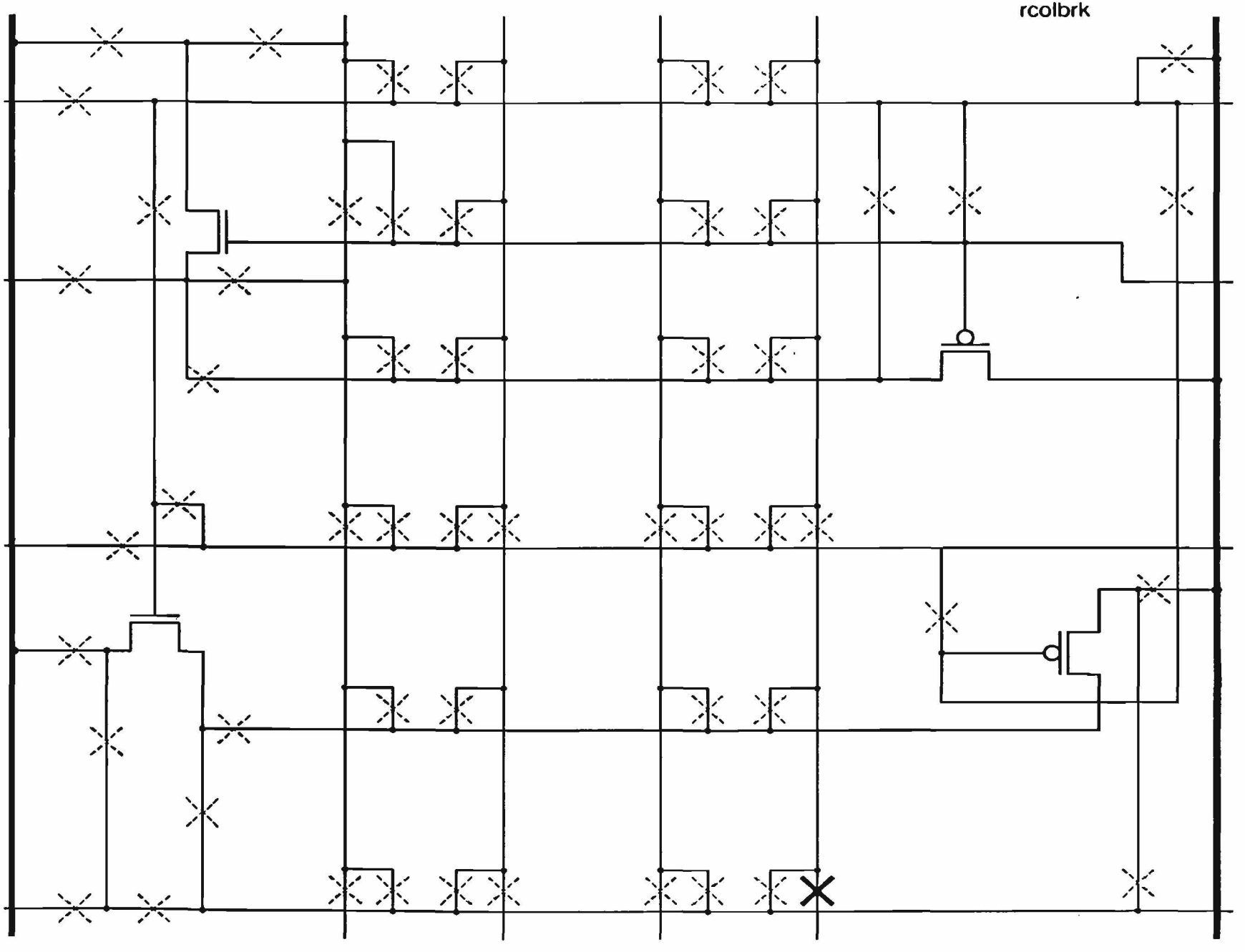
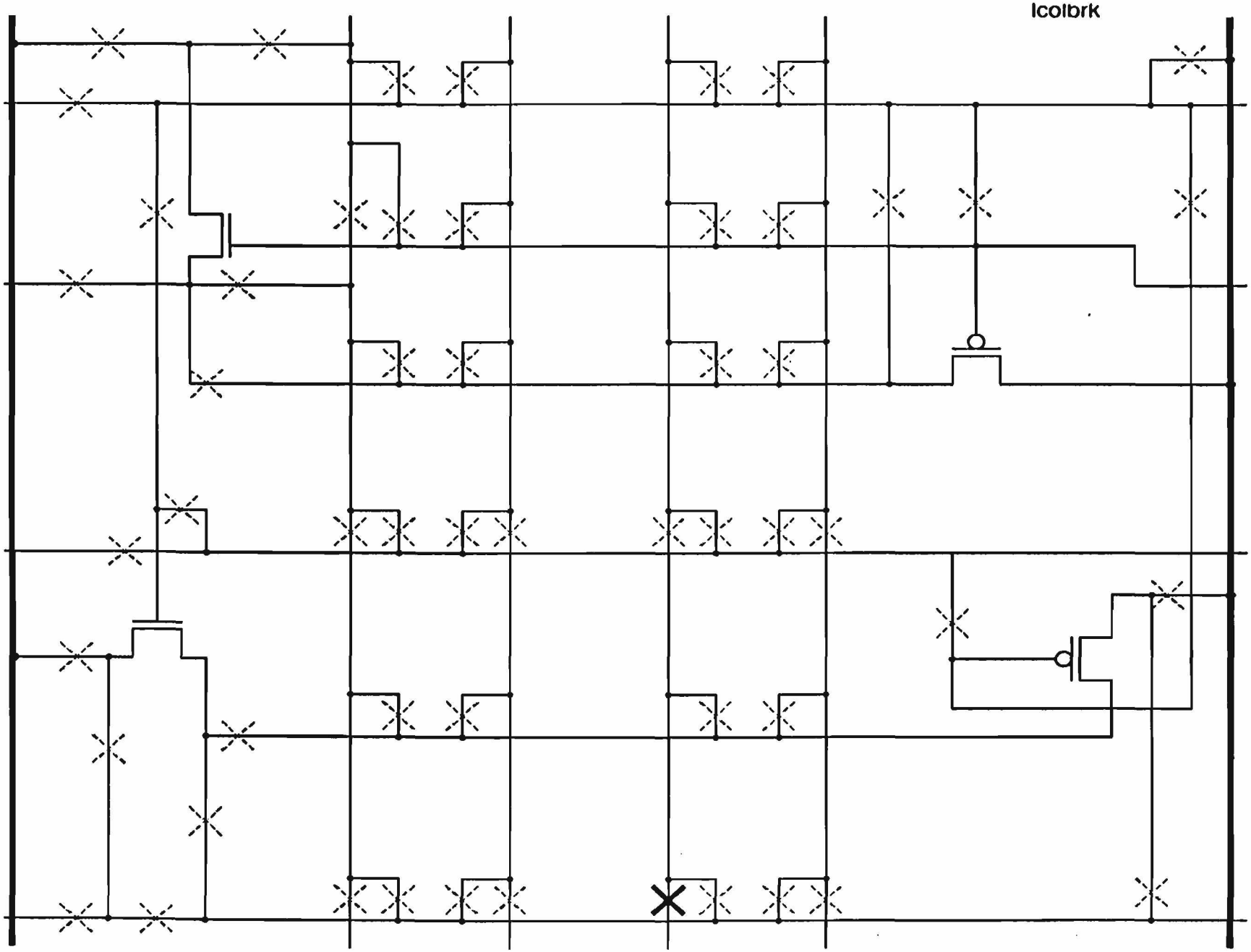
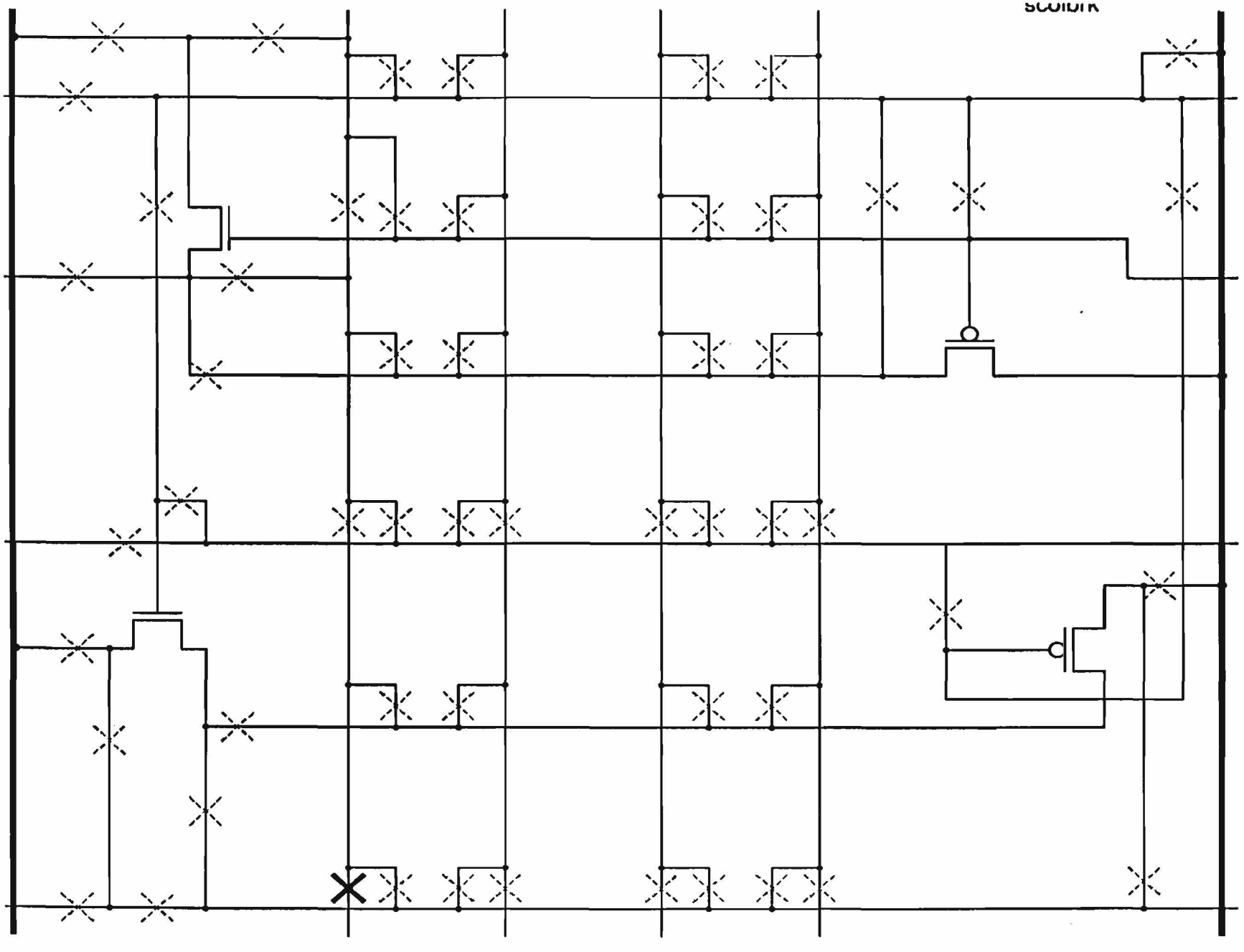
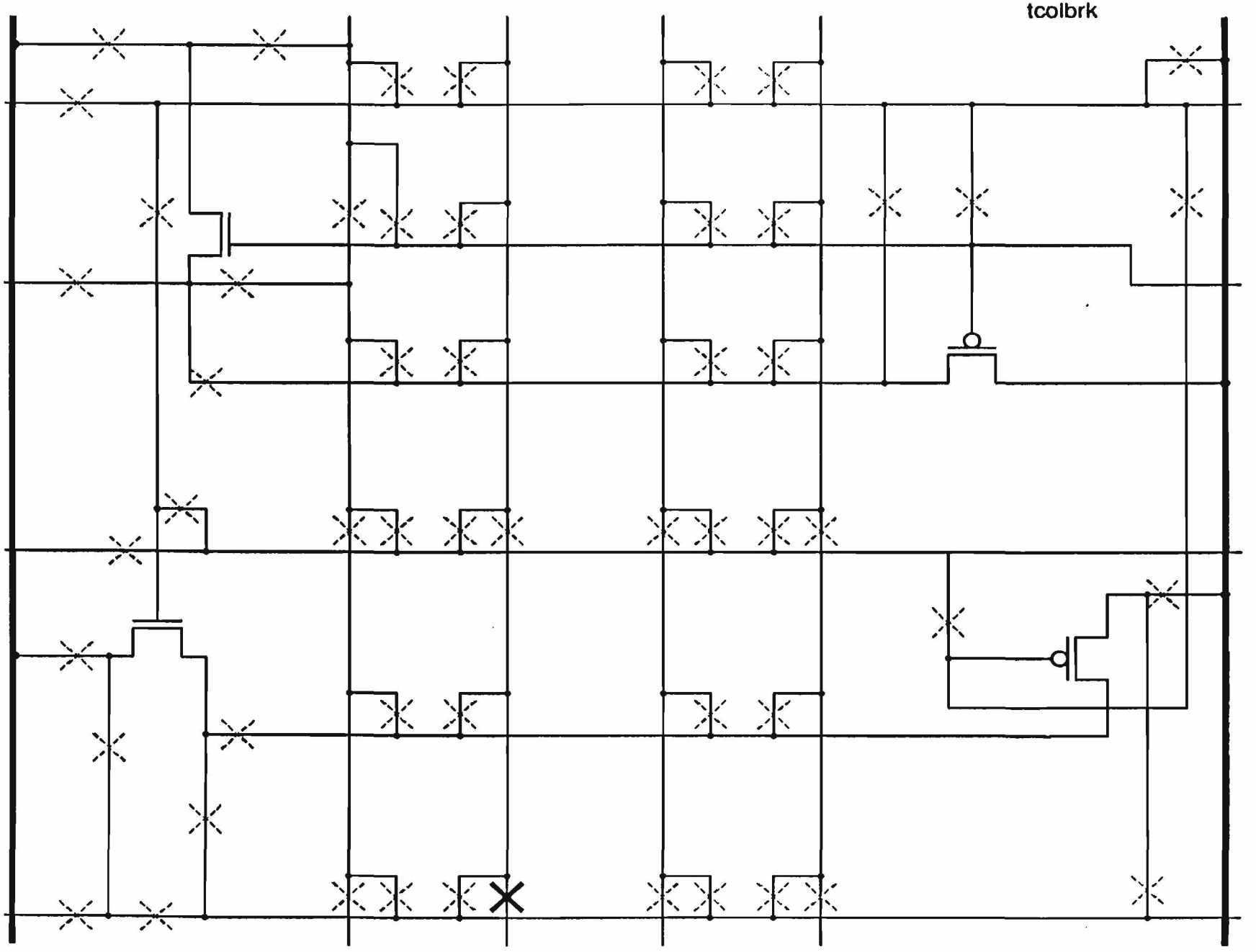


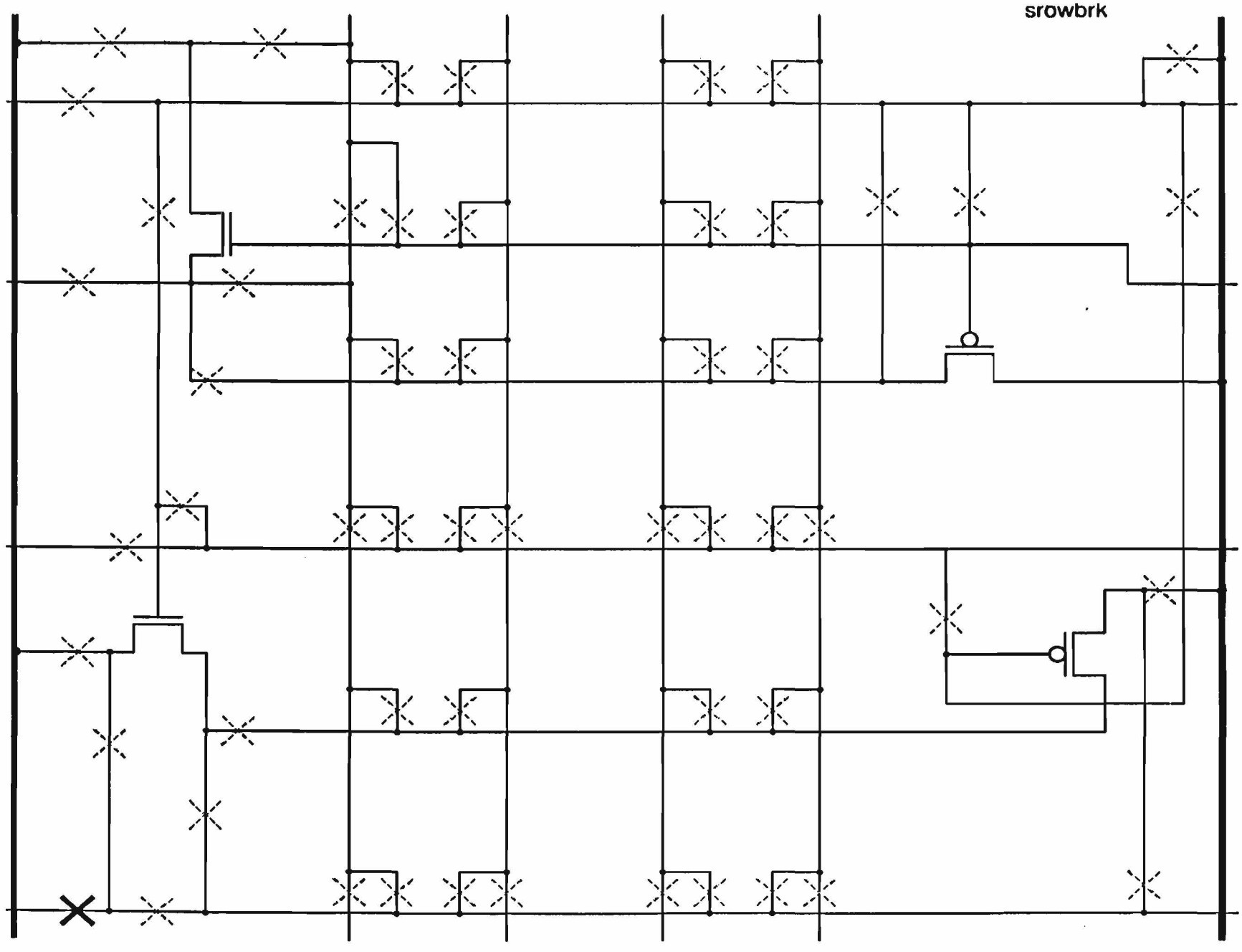
Figure 3.20. All of the possible breaks

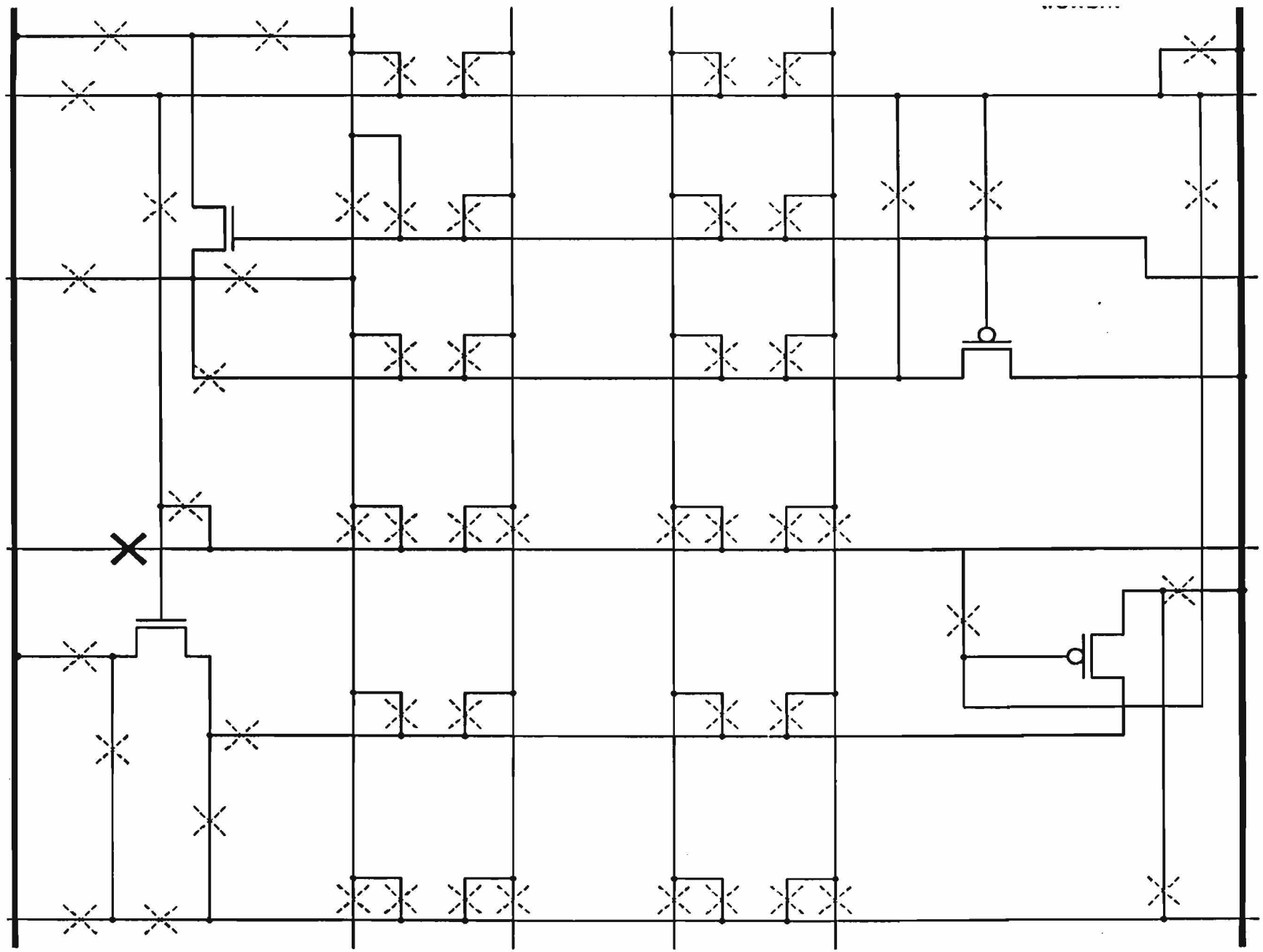

rcolbrk

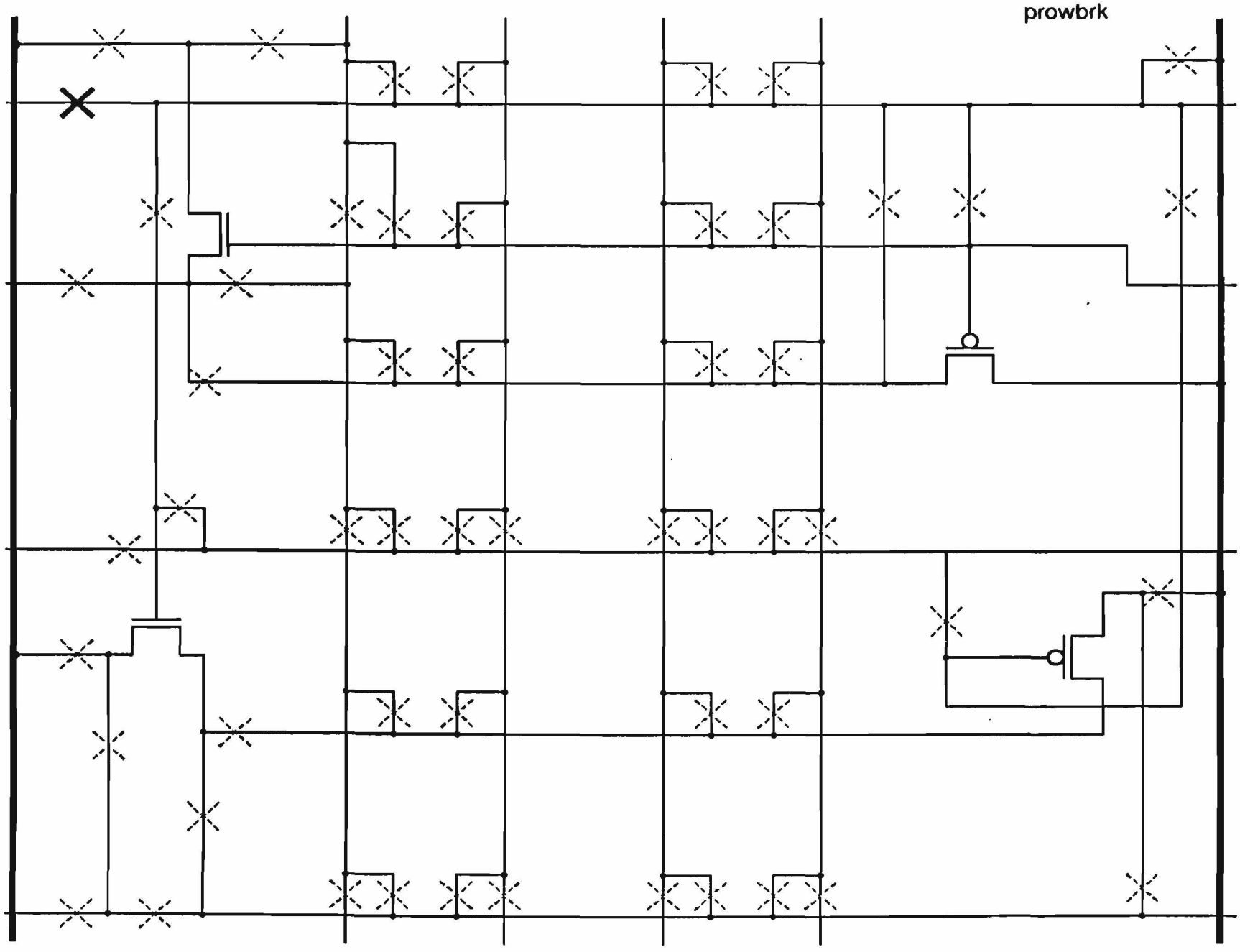

lcolbrk




tcolbrk


srowbrk




prowbrk

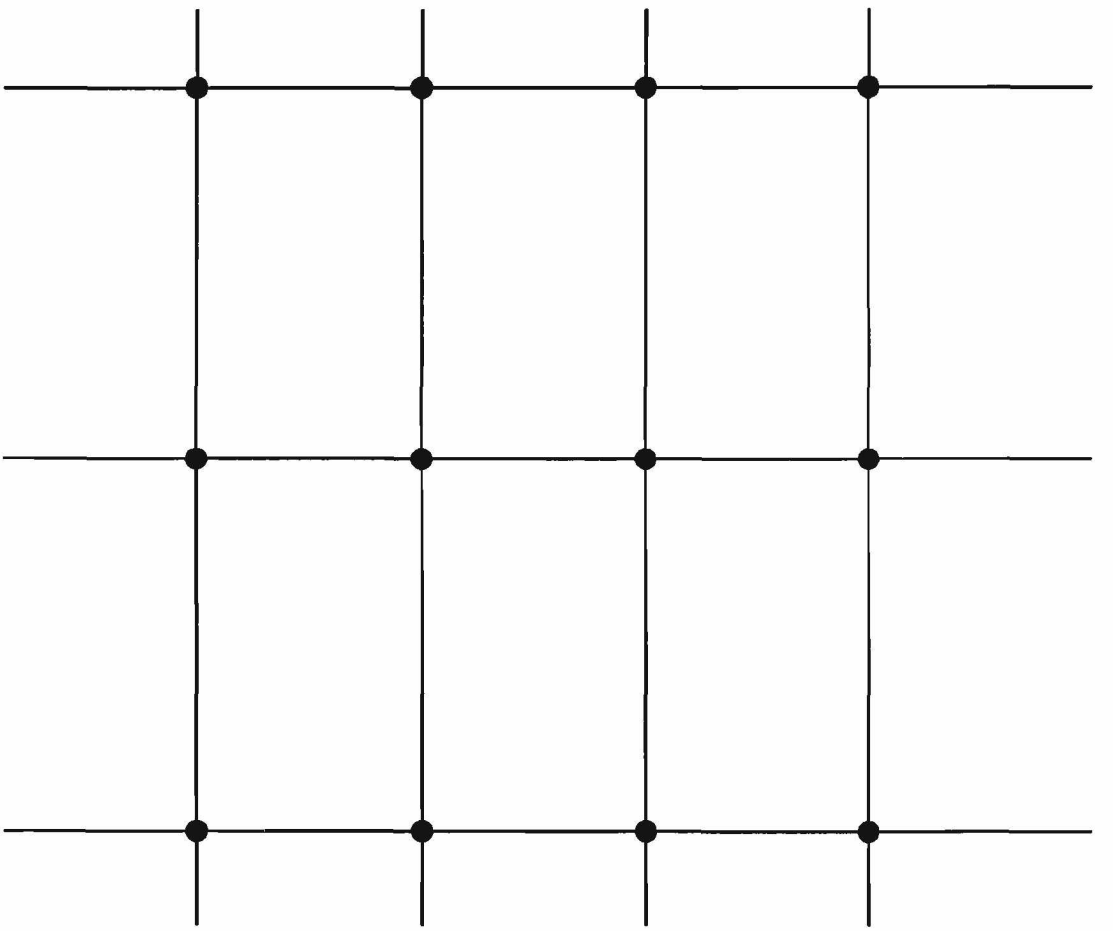
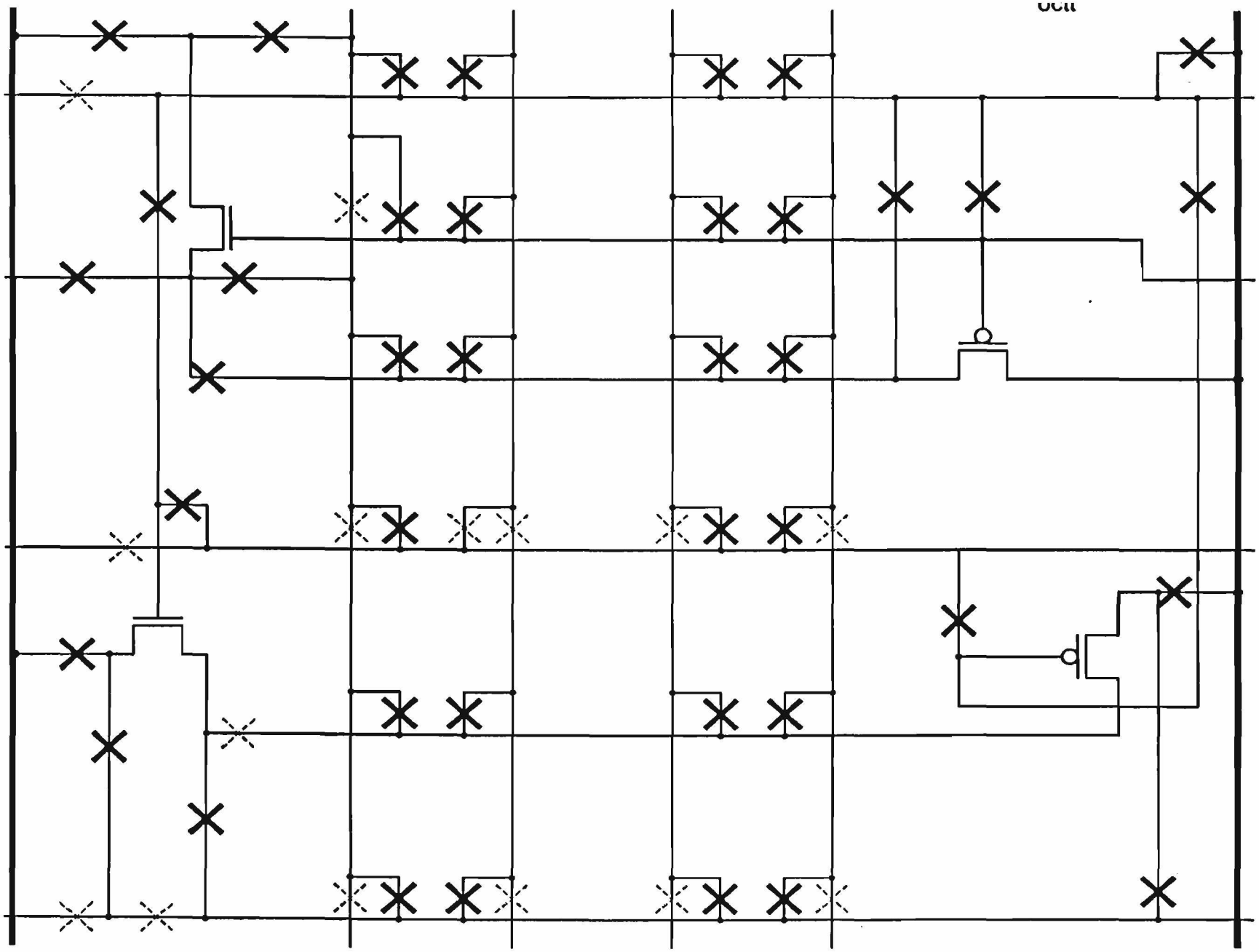
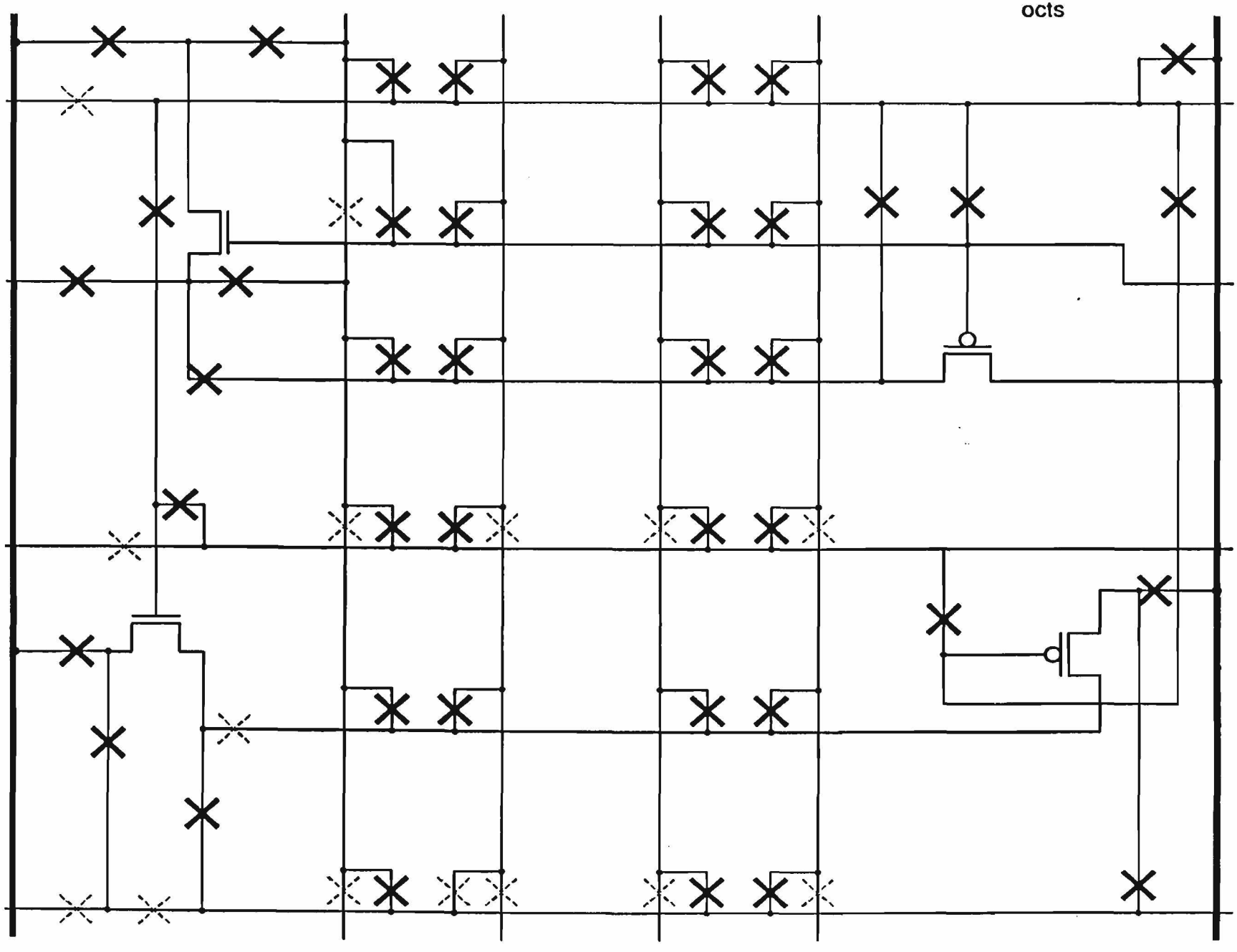
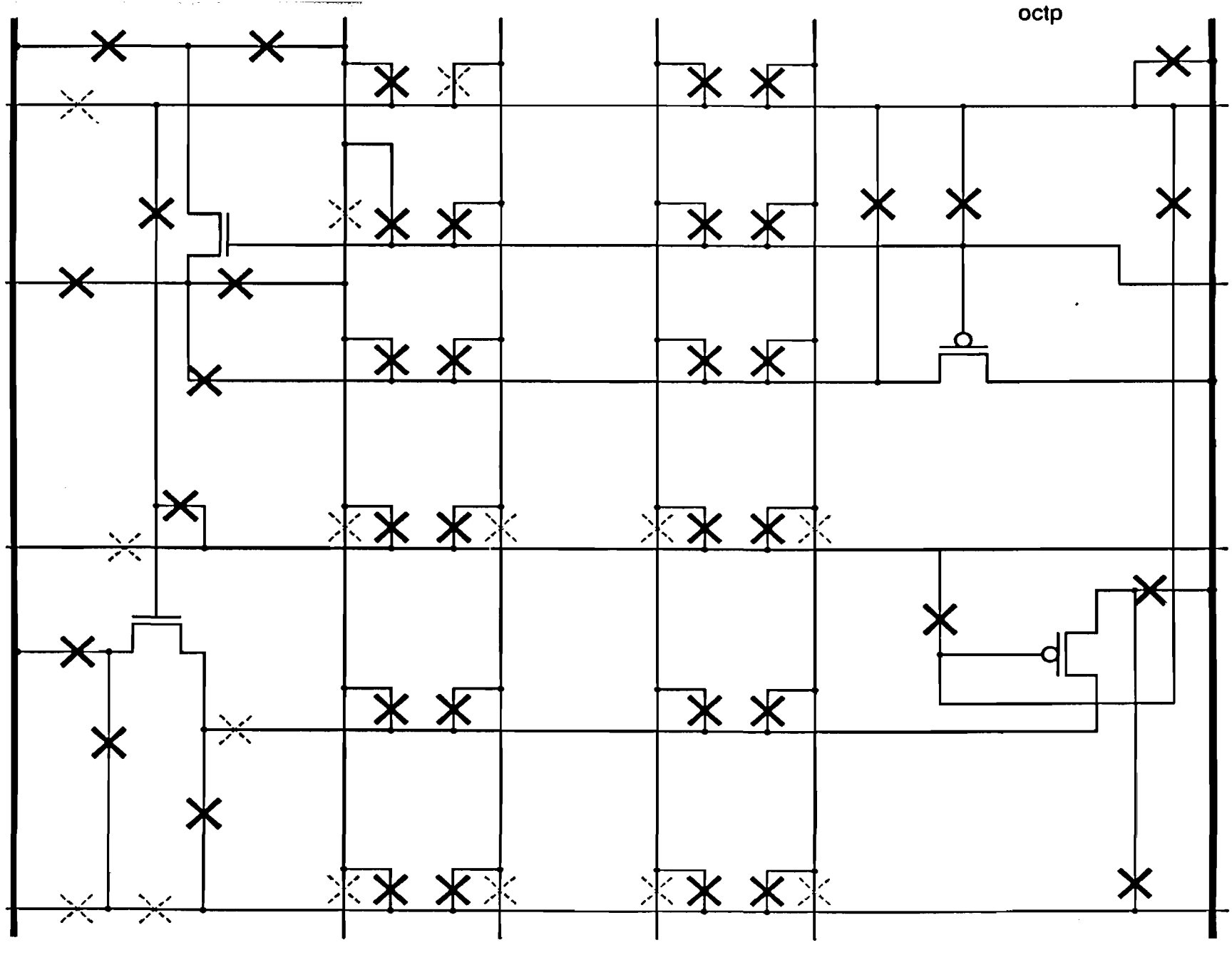
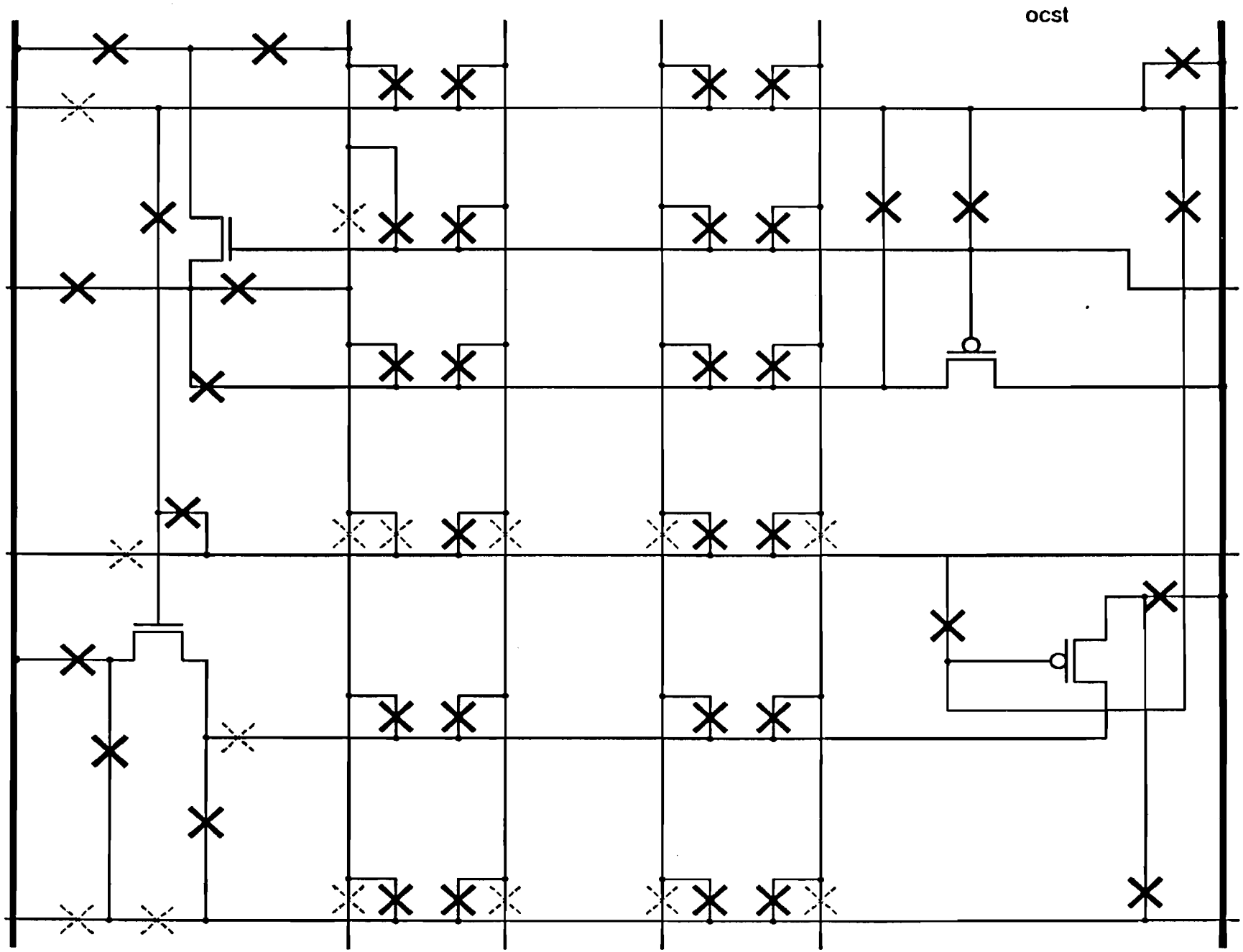


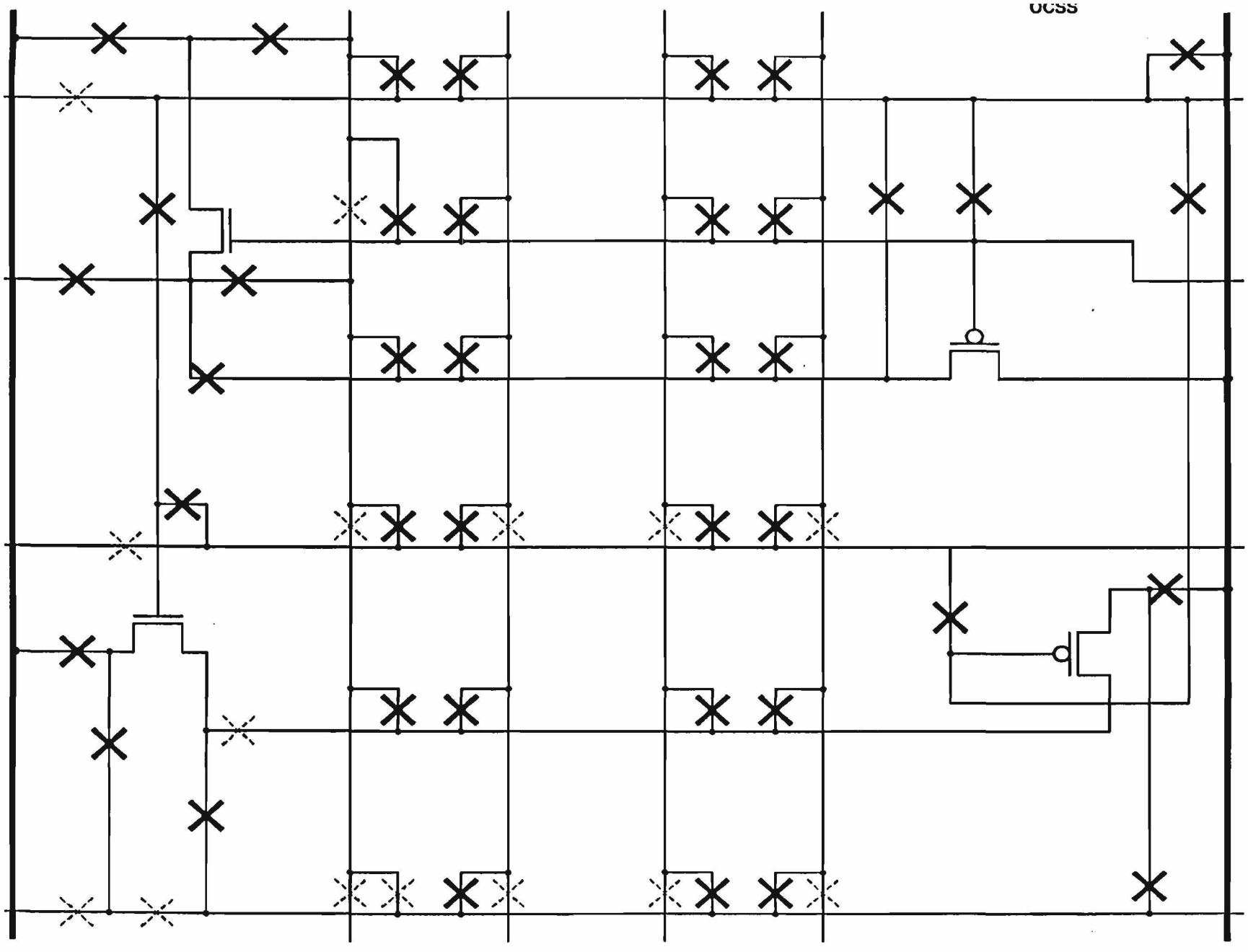
Figure 3.21. All possible connections

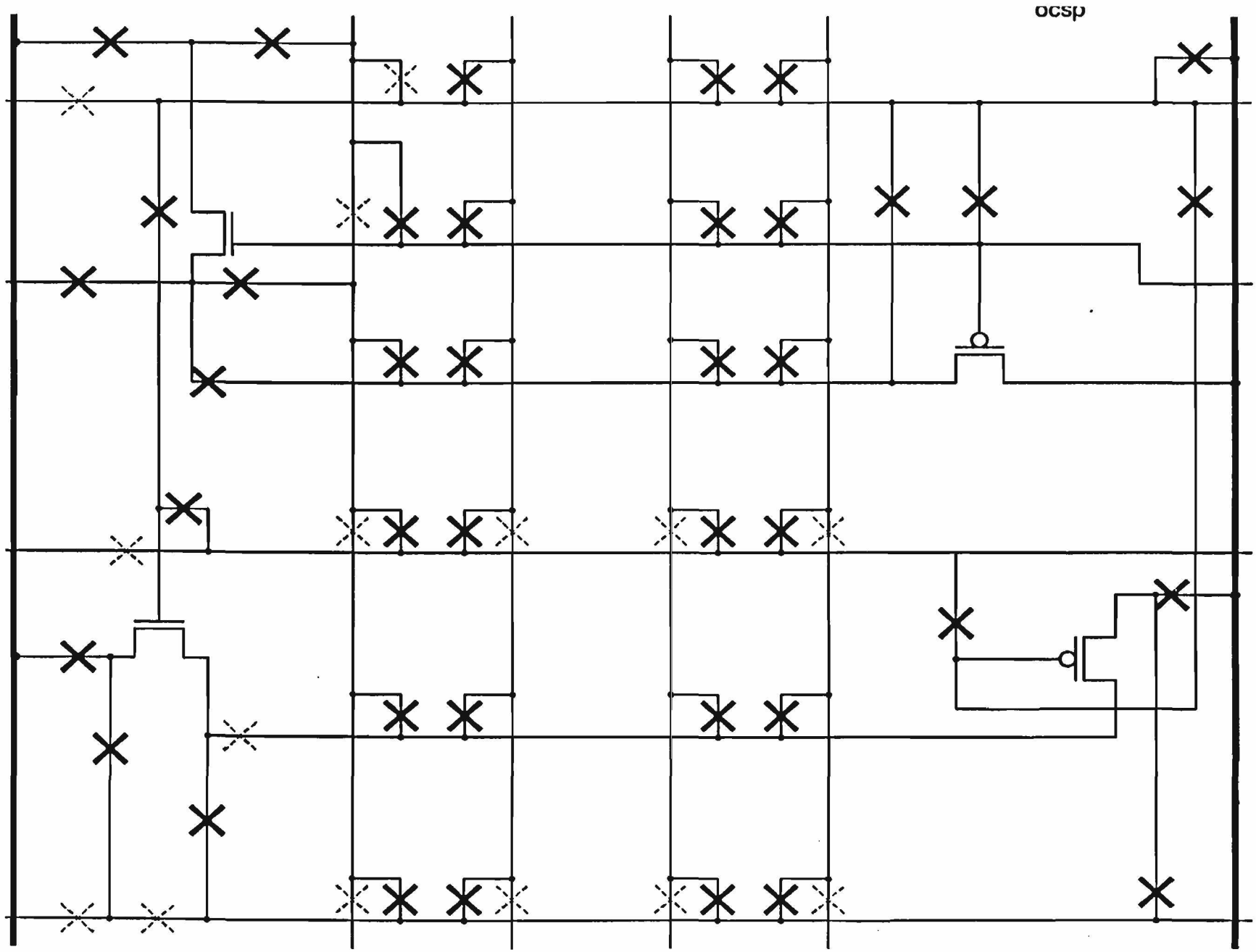


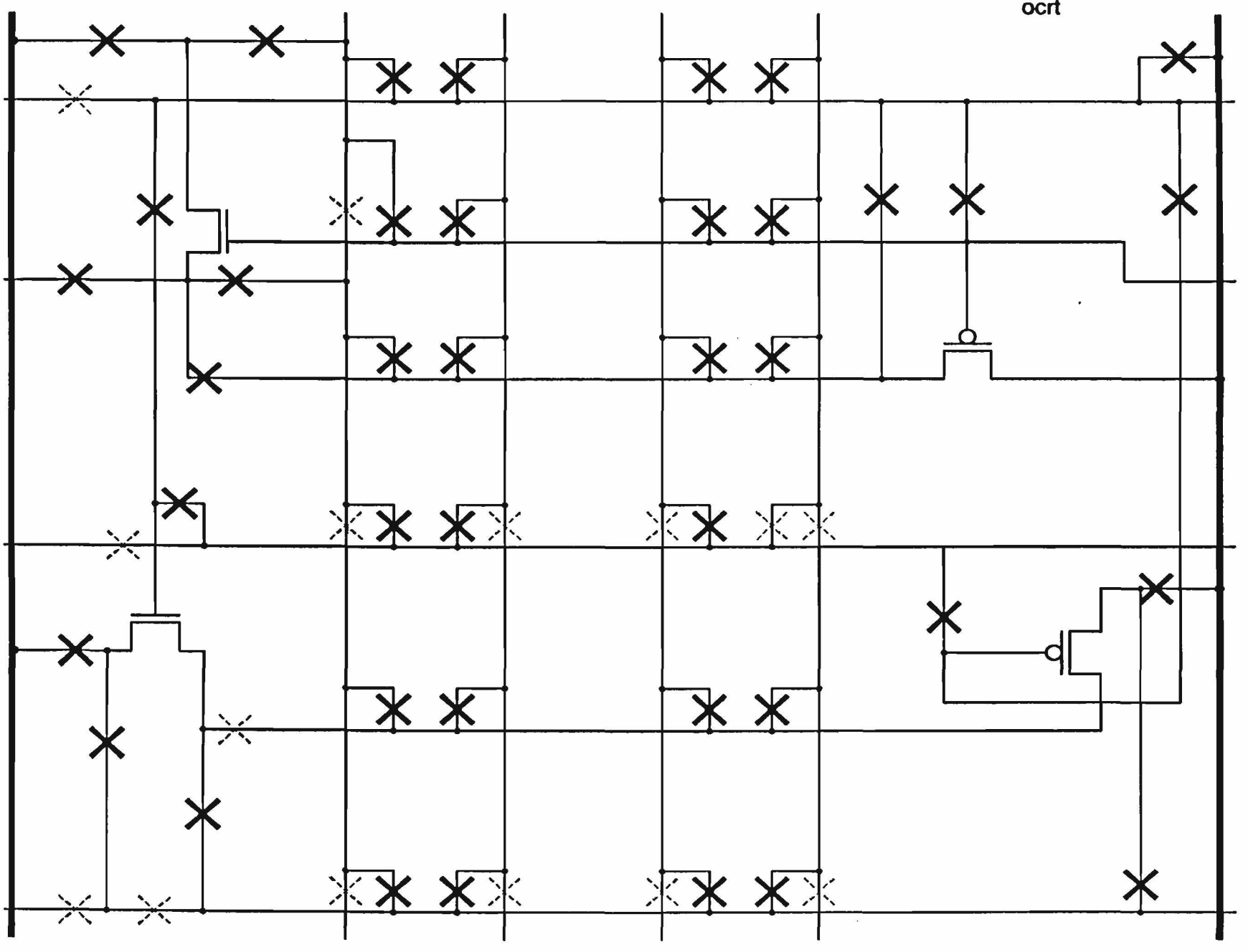

octs

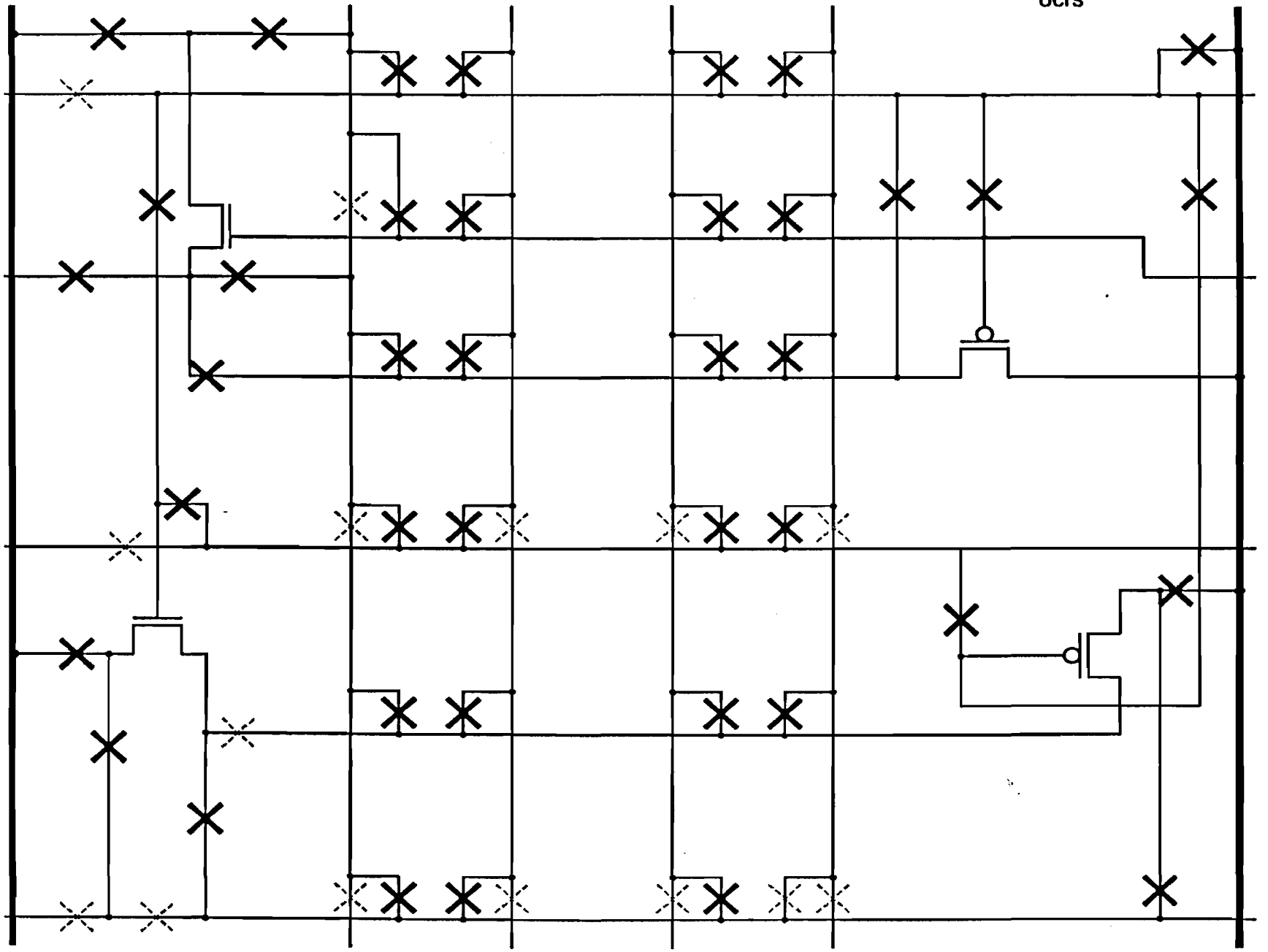

octp

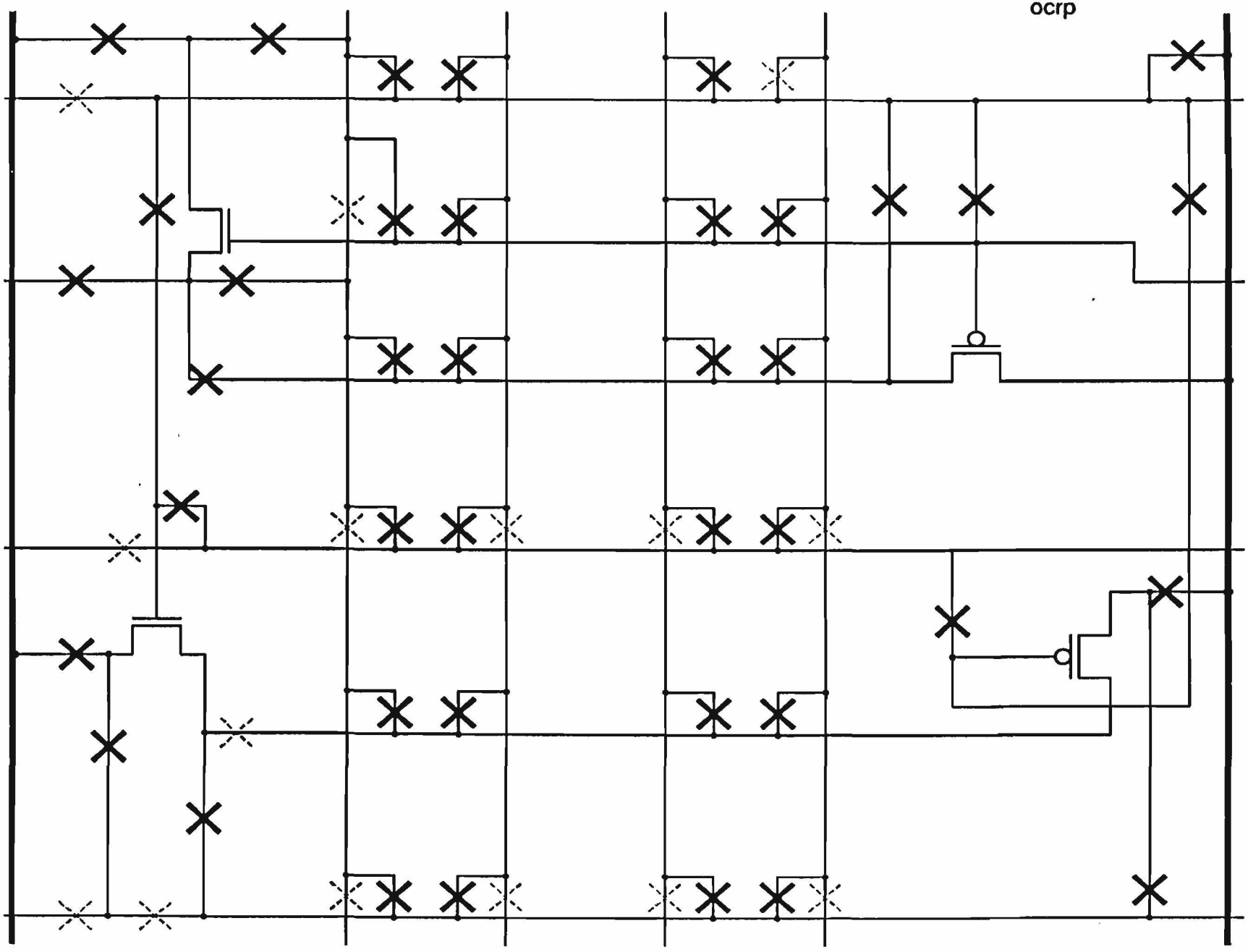

ocst

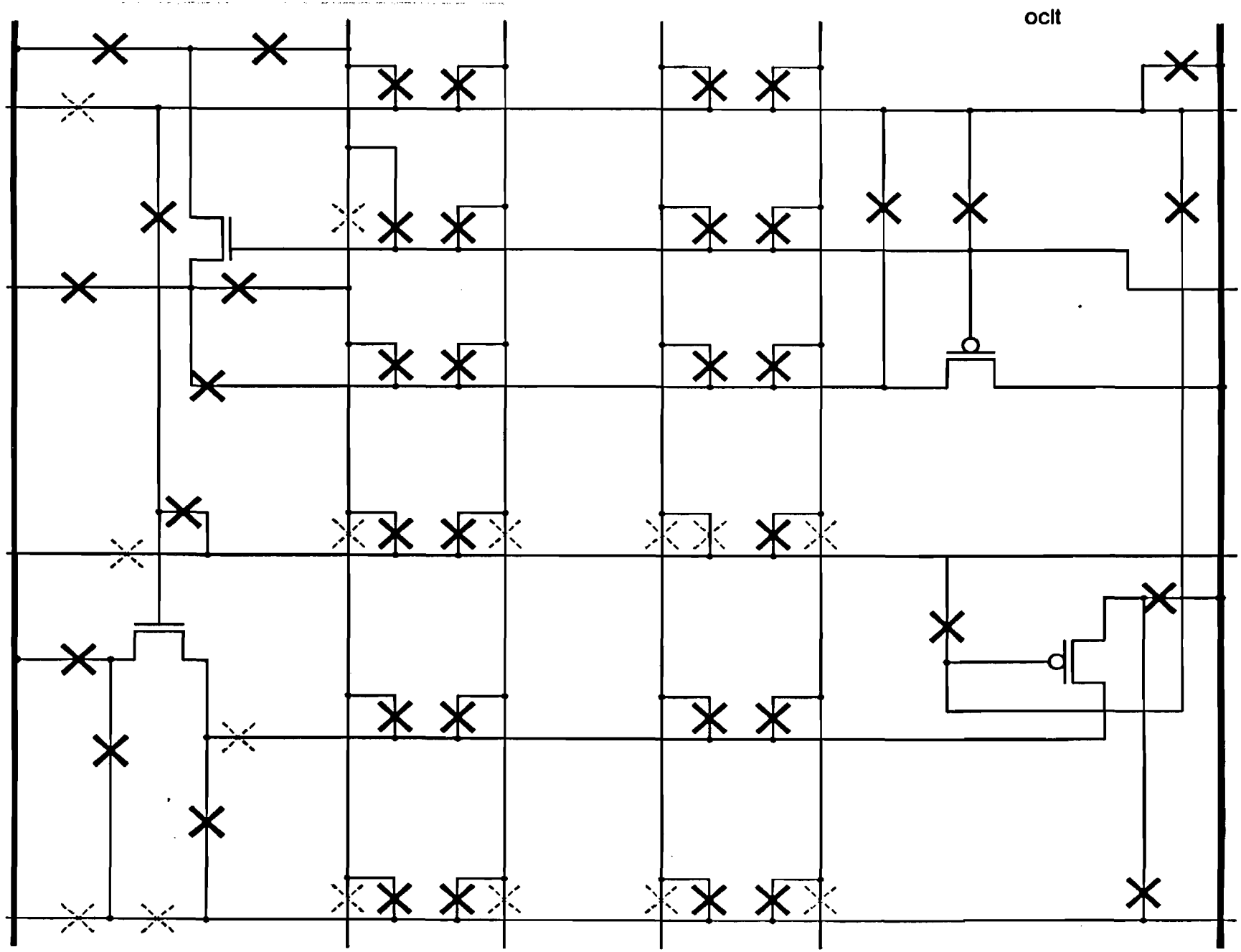

OCSS

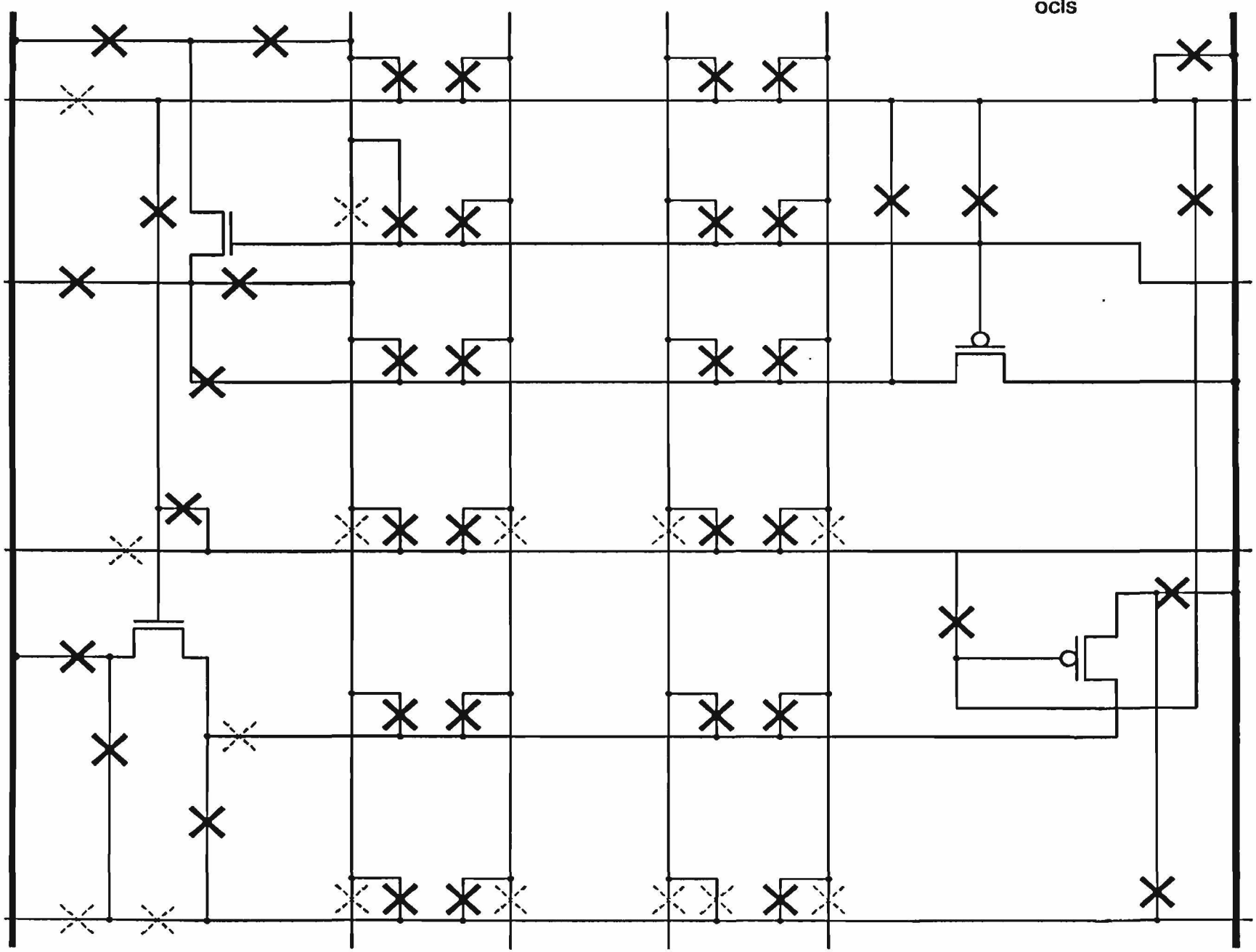

ocsp

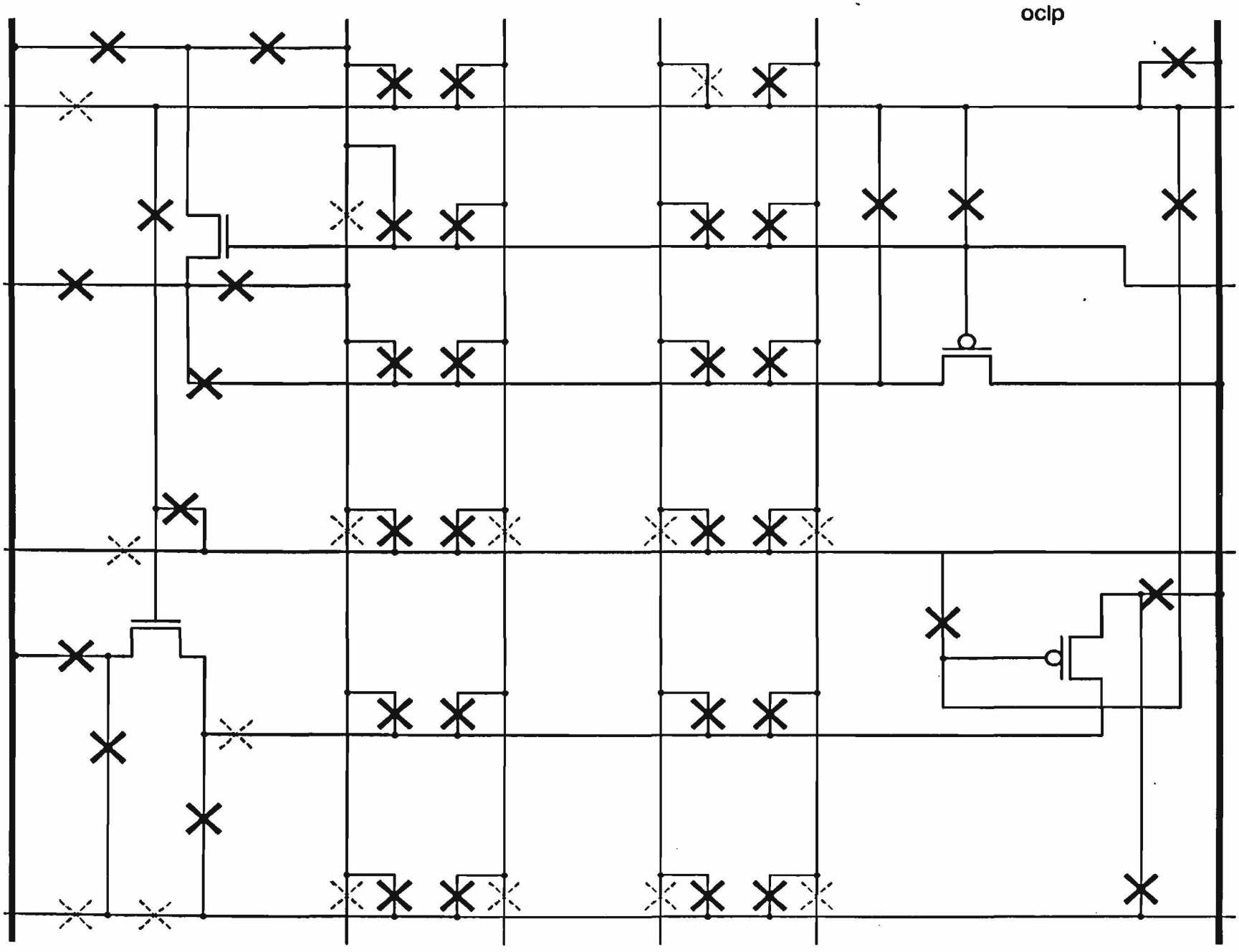

ocrt




ocrp


oclt


ocls


oclp

REFERENCES

[1] Jacobs, S. R., and Smith, K. F. PPL Quick Reference Guide. Tech. Rep. UUCS-87-002, University of Utah, August 1988.